A Survey of Deep Learning for Scientific Discovery

Maithra Raghu[1,2*] Eric Schmidt[1,3]
[1] Google
[2] Cornell University
[3] Schmidt Futures

Abstract

Over the past few years, we have seen fundamental breakthroughs in core problems in machine learning, largely driven by advances in deep neural networks. At the same time, the amount of data collected in a wide array of scientific domains is dramatically increasing in both size and complexity. Taken together, this suggests many exciting opportunities for deep learning applications in scientific settings. But a significant challenge to this is simply knowing where to start. The sheer breadth and diversity of different deep learning techniques makes it difficult to determine what scientific problems might be most amenable to these methods, or which specific combination of methods might offer the most promising first approach. In this survey, we focus on addressing this central issue, providing an overview of many widely used deep learning models, spanning visual, sequential and graph structured data, associated tasks and different training methods, along with techniques to use deep learning with less data and better interpret these complex models — two central considerations for many scientific use cases. We also include overviews of the full design process, implementation tips, and links to a plethora of tutorials, research summaries and open-sourced deep learning pipelines and pretrained models, developed by the community. We hope that this survey will help accelerate the use of deep learning across different scientific domains.

1 Introduction

The past few years have witnessed extraordinary advances in machine learning using deep neural networks. Driven by the rapid increase in available data and computational resources, these neural network models and algorithms have seen remarkable developments, and are a staple technique in tackling fundamental tasks ranging from speech recognition [70, 167], to complex tasks in computer vision such as image classification, (instance) segmentation, action recognition [117, 78, 240], and central problems in natural language, including question answering, machine translation and summarization [186, 172, 233, 197]. Many of these fundamental tasks (with appropriate reformulation) are relevant to a much broader array of domains, and in particular have tremendous potential in aiding the investigation of central scientific questions.

However, a significant obstacle in beginning to use deep learning is simply knowing where to start. The vast research literature, coupled with the enormous number of underlying models, tasks and training methods makes it very difficult to identify which techniques might be most appropriate to try, or the best way to start implementing them.

The goal of this survey is to help address this central challenge. In particular, it has the following attributes:

- The survey overviews a highly diverse set of deep learning concepts, from deep neural network models for varied data modalities (CNNs for visual data, graph neural networks, RNNs and Transformers for

*Correspondence to maithrar@gmail.com

sequential data) to the many different key tasks (image segmentation, super-resolution, sequence to sequence mappings and many others) to the multiple ways of training deep learning systems.

- But the explanation of these techniques is relatively high level and concise, to ensure the core ideas are accessible to a broad audience, and so that the entire survey can be read end to end easily.
- From the perspective of aiding scientific applications, the survey describes in detail (i) methods to use deep learning with less data (self-supervision, semi-supervised learning, and others) and (ii) techniques for interpretability and representation analysis (for going beyond predictive tasks). These are two exciting and rapidly developing research areas, and are also of particular significance to possible scientific use cases.
- The survey also focuses on helping quickly ramp up implementation, and in addition to overviews of the entire deep learning design process and a section on implementation tips (Section 9), the survey has a plethora of open-sourced code, research summaries and tutorial references developed by the community throughout the text, including a full section (Section 3) dedicated to this.

Who is this survey for? We hope this survey will be especially helpful for those with a basic understanding of machine learning, interested in (i) getting a comprehensive but accessible overview of many fundamental deep learning concepts and (ii) references and guidance in helping ramp up implementation. Beyond the core areas of deep learning, the survey focuses on methods to develop deep learning systems with less data, and techniques for interpreting these models, which we hope will be of particular use for those interested in applying these techniques in scientific problems. However, these topics and many others presented, along with the many code/tutorial/paper references may be helpful to anyone looking to learn about and implement deep learning.

1.1 Outline of Survey

The survey is structured as follows:

- Section 2 starts with some high level considerations for using deep learning. Specifically, we first discuss some template ways in which deep learning might be applied in scientific domains, followed by a general overview of the entire deep learning design process, and conclude with a brief discussion of other central machine learning techniques that may be better suited to some problems. The first part may be of particular interest to those considering scientific applications, while the latter two parts may be of general interest.
- Section 3 provides references to tutorials, open-sourced code model/algorithm implementations, and websites with research paper summaries, all developed by the deep learning community. This section should be very helpful for many readers and we encourage skimming through the links provided.
- Section 4 then overviews many of the standard tasks and models in deep learning, covering convolutional networks and their many uses, graph neural networks, sequence models (RNNs, Transformers) and the many associated sequence tasks.
- Section 5 looks at some key variants of the supervised learning training process, such as transfer learning, domain adaptation and multitask learning. These are central to many successful applications of deep learning.
- Section 6 considers ways to improve the data efficiency for developing deep neural network models, which has been a rapidly evolving area of research, and a core consideration for many applications, including scientific domains. It covers the many variants of self-supervision and semi-supervised learning, as well as data augmentation and data denoising.

- Section 7 overviews advances in interpretability and representational analysis, a set of techniques focused on gaining insights into the internals of the end-to-end system: identifying important features in the data, understanding its effect on model outputs and discovering properties of model hidden representations. These are very important for many scientific problems which emphasise understanding over predictive accuracy, and may be of broader interest for e.g. aiding model debugging and preemptively identifying failure modes.
- Section 8 provides a brief overview of more advanced deep learning methods, specifically generative modelling and reinforcement learning.
- Section 9 concludes with some key implementation tips when putting together an end-to-end deep learning system, which we encourage a quick read through!

2 High Level Considerations for Deep Learning

In this section we first discuss some high level considerations for deep learning techniques. We start with overviews of template ways in which deep learning might be applied in scientific settings, followed by a discussion of the end-to-end design process and some brief highlights of alternate machine learning methods which may be more suited to some problems.

2.1 Templates for Deep Learning in Scientific Settings

What are the general ways in which we might apply deep learning techniques in scientific settings? At a very high level, we can offer a few *templates* of ways in which deep learning might be used in such problems:

(1) *Prediction Problems* Arguably the most straightforward way to apply deep learning is to use it to tackle important *prediction problems*: mapping inputs to predicted outputs. This predictive use case of deep learning is typically how it is also used in core problems in computing and machine learning. For example, the input might be a biopsy image, and the model must output a prediction of whether the imaged tissue shows signs of cancer. We can also think of this predictive use case as getting the model to *learn a target function*, in our example, mapping from input visual features to the cancer/no cancer output. Using deep learning in this way also encapsulates settings where the target function is very complex, with no mathematical closed form or logical set of rules that describe how to go from input to output. For instance, we might use a deep learning model to (black-box) *simulate* a complex process (e.g. climate modelling), that is very challenging to explicitly model [101].

(2) *From Predictions to Understanding* One fundamental difference between scientific questions and core machine learning problems is the emphasis in the former on *understanding* the underlying mechanisms. Oftentimes, outputting an accurate prediction alone is not enough. Instead, we want to gain interpretable insights into what properties of the data or the data generative process led to the observed prediction or outcome. To gain these kinds of insights, we can turn to interpretability and representation analysis methods in deep learning, which focus on determining how the neural network model makes a specific prediction. There has been significant work on both tools to understand what features of the input are most critical to the output prediction, as well as techniques to directly analyze the *hidden representations* of the neural network models, which can reveal important properties of the underlying data.

(3) *Complex Transformations of Input Data* In many scientific domains, the amount of generated data, particularly visual data (e.g. fluorescence microscopy, spatial sequencing, specimen videos [177, 97]) has grown dramatically, and there is an urgent need for efficient analysis and automated processing. Deep learning techniques, which are capable of many complex transformations of data, can be highly effective for such settings, for example, using a deep neural network based segmentation model to automatically

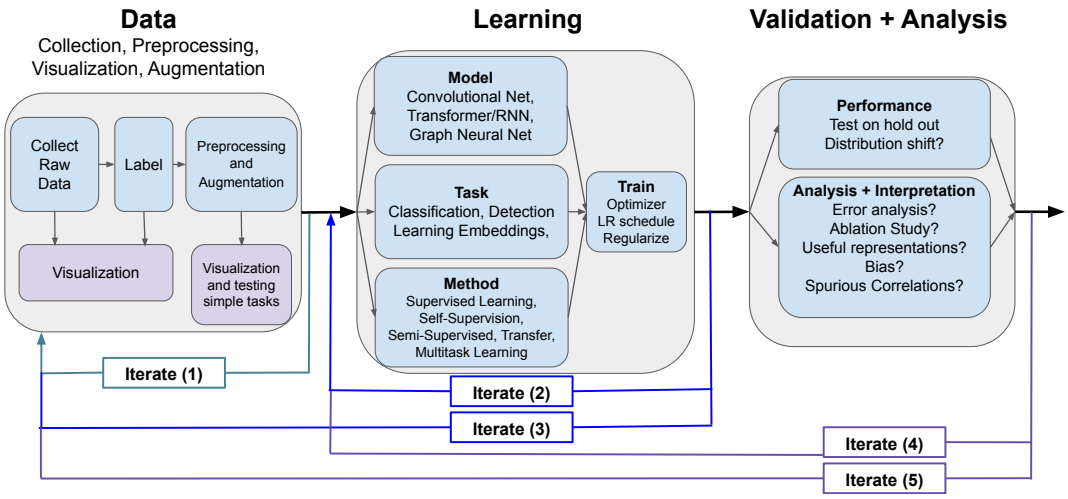

Figure 1: **Schematic of a typical deep learning workflow.** A typical development process for deep learning applications can be viewed as consisting of three sequential stages (i) data related steps (ii) the learning component (iii) validation and analysis. Each one of these stages has several substeps and techniques associated with it, also depicted in the figure. In the survey we will overview most techniques in the learning component, as well as some techniques in the data and validation stages. Note that while a natural sequence is to first complete steps in the data stage, followed by learning and then validation, standard development will likely result in multiple different iterations where the techniques used or choices made in one stage are revisited based off of results of a later stage.

identify the nuclei in images of cells, or a pose estimation system to rapidly label behaviors seen in videos of mice for neuroscience analysis.

2.2 Deep Learning Workflow

With these examples of templates for deep learning applications in science, we next look at the end to end workflow for designing a deep learning system. Figure 1 illustrates what a typical workflow might look like.

Having selected the overarching (predictive) problem of interest, we can broadly think of having three stages for designing and using the deep learning system: (i) *data* related steps, such as collection, labelling, preprocessing, visualization, etc (ii) *learning* focused steps, such as choice of deep neural network model, the task and method used to train the model (iii) *validation and analysis* steps, where performance evaluations are conducted on held out data, as well as analysis and interpretation of hidden representations and ablation studies of the overall methods.

These three stages are naturally sequential. However, almost all of the time, the first attempt at building an end-to-end deep learning system will result in some kind of failure mode. To address these, it is important to keep in mind the *iterative* nature of the design process, with results from the different stages informing the redesign and rerunning of other stages.

Figure 1 provides some examples of common iterations with the backward connecting arrows: (i) the *Iterate (1)* arrow, corresponding to iterations on the data collection process, e.g. having performed some data visualization, the labelling process for the raw instances might require adjusting — the first labelling mechanism might be too noisy, or not capture the objective of interest (ii) the *Iterate (2)* arrow, corresponding to iterations on the learning setup, due to e.g. deciding that a different task or method might be more appropriate, or decomposing the learning process into multiple steps — first performing self-supervision followed by supervised learning (iii) the *Iterate (3)* arrow, changing the data related steps based off of the results of the learning step (iv) the *Iterate (4)* arrow, redesigning the learning process informed by the

validation results e.g. finding out the model has overfit on the training data at validation and hence reducing training time or using a simpler model (v) the *Iterate (5)* arrow, adapting the data steps based off the validation/analysis results, e.g. finding that the model is relying on spurious attributes of the data, and improving data collection/curation to mitigate this.

Focus of Survey and Nomenclature In this survey, we provide a comprehensive overview of many of the techniques in the *learning stage*, along with some techniques (e.g. data augmentation, interpretability and representation analysis, Section 7) in the data and validation stages.

For the learning stage, we look at popular *models*, *tasks* and *methods*. By *models* (also sometimes referred to as *architecture*), we mean the actual structure of the deep neural network — how many layers, of what type, and how many neurons, etc. By *tasks*, we mean the kind of prediction problem, specifically, the type of input and output. For example, in an *image classification* task, the input consists of images and the output a probability distribution over a (discrete) set of different categories (called classes). By *methods*, we refer to the type of learning process used to train the system. For example, *supervised learning* is a very general learning process, consisting of the neural network being given data instances with corresponding *labels*, with the labels providing supervision.

Unlike different models and tasks, methods can be subsets of other methods. For example, *self-supervision*, a method where the neural network is trained on data instances and labels, but the labels automatically created from the data instance, can also be considered a type of *supervised learning*. This can be a little confusing! But it suffices to keep in mind the general notions of *models*, *tasks* and *methods*.

2.3 Deep Learning or Not?

As a final note before diving into the different deep learning techniques, when formulating a problem, it is important to consider whether deep learning provides the right set of tools to solve it. The powerful underlying neural network models offer many sophisticated functionalities, such learned complex image transforms. However, in many settings, deep learning may not be the best technique to start with or best suited to the problem. Below we very briefly overview some of the most ubiquitous machine learning methods, particularly in scientific contexts.

Dimensionality Reduction and Clustering In scientific settings, the ultimate goal of data analysis is often *understanding* — identifying the underlying mechanisms that give rise to patterns in the data. When this is the goal, *dimensionality reduction*, and/or *clustering* are simple (unsupervised) but highly effective methods to reveal hidden properties in the data. They are often very useful in the important first step of exploring and visualizing the data (even if more complex methods are applied later.)

Dimensionality Reduction: Dimensionality reduction methods are either *linear*, relying on a linear transformation to reduce data dimensionality, or *non-linear*, reducing dimensionality while approximately preserving the non-linear (manifold) structure of the data. Popular linear dimensionality reduction methods include *PCA* and *non-negative matrix factorization*, with some popular non-linear methods including *t-SNE* [141] and *UMAP* [148]. Most dimensionality reduction methods have high-quality implementations in packages like `scikit-learn` or on github, e.g. https://github.com/oreillymedia/t-SNE-tutorial or https://github.com/lmcinnes/umap.

Clustering: Often used in combination with dimensionality reduction, clustering methods provide a powerful, unsupervised way to identify similarities and differences across the data population. Commonly used clustering methods include *k-means* (particularly the *k-means++* variant), *Gaussian Mixture Models* (GMMs), *hierarchical clustering* and *spectral clustering*. Like dimensionality reduction techniques, these clustering methods have robust implementations in packages like `scikit-learn`.

In Section 7.2.2, we discuss how dimensionality reduction and clustering can be used on the hidden representations of neural networks.

Linear Regression, Logistic Regression (and variants!) Arguably the most fundamental techniques for *supervised* problems like classification and regression, linear and logistic regression, and their variants (e.g. Lasso, Ridge Regression) may be particularly useful when there is limited data, and a clear set of (possibly preprocessed) features (such as in tabular data.) These methods also often provide a good way to sanity check the overarching problem formulation, and may be a good starting point to test out a very simple version of the full problem. Due to their simplicity, linear and logistic regression are highly interpretable, and provide straightforward ways to perform *feature attribution*.

Decision Trees, Random Forests and Gradient Boosting Another popular class of methods are decision trees, random forests and gradient boosting. These methods can also work with regression/classification tasks, and are well suited to model non-linear relations between the input features and output predictions. Random forests, which ensemble decision trees, can often be preferred to deep learning methods in settings where the data has a low signal-to-noise ratio. These methods can typically be less interpretable than linear/logistic regression, but recent work [160] has looked at developing software libraries `https://github.com/interpretml/interpret` to address this challenge.

Other Methods and Resources: Both the aforementioned techniques and many other popular methods such as graphical models, Gaussian processes, Bayesian optimization are overviewed in detail in excellent course notes such as University of Toronto's Machine Learning Course or Stanford's CS229, detailed articles at `https://towardsdatascience.com/` and even interactive textbooks such as `https://d2l.ai/index.html` (called Dive into Deep Learning [267]) and `https://github.com/rasbt/python-machine-learning-book-2nd-edition`.

3 Deep Learning Libraries and Resources

A remarkable aspect of advances in deep learning so far is the enormous number of resources developed and shared by the community. These range from tutorials, to overviews of research papers, to open sourced code. Throughout this survey, we will reference some of these materials in the topic specific sections, but we first list here a few general very useful frameworks and resources.

Software Libraries for Deep Learning: Arguably the two most popular code libraries for deep learning are PyTorch (with a high level API called Lightning) and TensorFlow (which also offers Keras as a high level API.) Developing and training deep neural network models critically relies on fast, parallelized matrix and tensor operations (sped up through the use of Graphical Processing Units) and performing automatic differentiation for computing gradients and optimization (known as *autodiff*.) Both PyTorch and TensorFlow offer these core utilities, as well as many other functions. Other frameworks include Chainer, ONNX, MXNET and JAX. Choosing the best framework has been the source of significant debate. For ramping up quickly, programming experiences closest to native Python, and being able to use many existing code repositories, PyTorch (or TensorFlow with the Keras API) may be two of the best choices.

Tutorials: (i) `https://course.fast.ai/` fast.ai provides a free, coding-first course on the most important deep learning techniques as well as an intuitive and easy to use code library, `https://github.com/fastai/fastai`, for model design and development. (ii) `https://towardsdatascience.com/` contains some fantastic tutorials on almost every deep learning topic imaginable, crowd sourced from many contributors. (iii)

Many graduate deep learning courses have excellent videos and lecture notes available online, such as http://www.cs.toronto.edu/~rgrosse/courses/csc421_2019/ for Deep Learning and Neural Networks, or the more topic specific Stanford's CS224N NLP with Deep Learning. A nice collection of some of these topic specific lectures is provided at https://github.com/Machine-Learning-Tokyo/AI_Curriculum. There are also some basic interactive deep learning courses online, such as https://github.com/leriomaggio/deep-learning-keras-tensorflow.

Research Overviews, Code, Discussion: (i) https://paperswithcode.com/ This excellent site keeps track of new research papers and their corresponding opensourced code, trending directions and displays state of the art results (https://paperswithcode.com/sota) across many standard benchmarks. (ii) Discussion of deep learning research is very active on Twitter. http://www.arxiv-sanity.com/top keeps track of some of the top most discussed papers and comments. (iii) https://www.reddit.com/r/MachineLearning/ is also a good forum for research and general project discussion. (iv) https://www.paperdigest.org/conference-paper-digest/ contains snippets of all the papers in many different top machine learning conferences. (v) IPAM (Institute for Pure and Applied Mathematics) has a few programs e.g. https://www.ipam.ucla.edu/programs/workshops/new-deep-learning-techniques/?tab=schedule and https://www.ipam.ucla.edu/programs/workshops/deep-learning-and-medical-applications/?tab=schedule with videos overviewing deep learning applications in science.

Models, Training Code and Pretrained Models: As we discuss later in the survey, publicly available models, training code and *pretrained* models are very useful for techniques such as transfer learning. There are many good sources of these, here are a few that are especially comprehensive and/or accessible:

(i) Pytorch and TensorFlow have a collection of pretrained models, found at https://github.com/tensorflow/models and https://pytorch.org/docs/stable/torchvision/models.html.

(ii) https://github.com/huggingface Hugging Face (yes, that really is the name), offers a huge collection of both pretrained neural networks and the code used to train them. Particularly impressive is their library of *Transformer* models, a one-stop-shop for sequential or language applications.

(iii) https://github.com/rasbt/deeplearning-models offers many standard neural network architectures, including multilayer perceptrons, convolutional neural networks, GANs and Recurrent Neural Networks.

(iv) https://github.com/hysts/pytorch_image_classification does a deep dive into image classification architectures, with training code, highly popular data augmentation techniques such as *cutout*, and careful speed and accuracy benchmarking. See their page for some object detection architectures also.

(v) https://github.com/openai/baselines provides implementations of many popular RL algorithms.

(vi) https://modelzoo.co/ is a little like paperswithcode, but for models, linking to implementations of neural network architectures for many different standard problems.

(vii) https://github.com/rusty1s/pytorch_geometric. Implementations and paper links for many graph neural network architectures.

Data Collection, Curation and Labelling Resources: A crucial step in applying deep learning to a problem is collecting, curating and labelling data. This is a very important, time-intensive and often highly intricate task (e.g. labelling object boundaries in an image for segmentation.) Luckily, there are some resources and libraries to help with this, for example https://github.com/tzutalin/labelImg, https://github.com/wkentaro/labelme, https://rectlabel.com/ for images and https://github.com/doccano/doccano for text/sequential data.

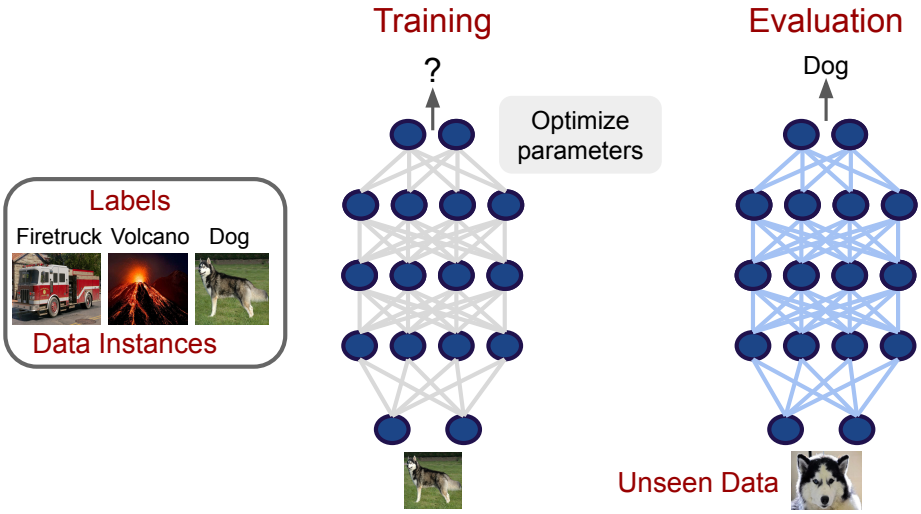

Figure 2: **The Supervised Learning process for training neural networks.** The figure illustrates the supervised learning process for neural networks. Data instances (in this case images) and corresponding labels are collected. During the training step, the parameters of the neural network are optimized so that when input a data instance, the neural network outputs the corresponding label. During evaluation, the neural network is given unseen data instances as input, and if trained successfully, will output a meaningful label (prediction).

Visualization, Analysis and Compute Resources: When training deep neural network models, it is critical to visualize important metrics such as loss and accuracy *while* the model is training. Tensorboard `https://www.tensorflow.org/tensorboard` (which works with Pytorch and TensorFlow) is a very popular framework for doing this. Related is the *colab* effort `https://colab.research.google.com/notebooks/welcome.ipynb`, which, aside from providing a user-friendly, interactive way for model development and analysis (very similar to jupyter notebooks) also provides some (free!) compute resources.

4 Standard Neural Network Models and Tasks

In this section, we overview the standard *neural network models* and the kinds of *tasks* they can be used for, from convolutional networks for image predictions and transformations to transformer models for sequential data to graph neural networks for chemistry applications.

4.1 Supervised Learning

Before diving into the details of the different deep neural network models, it is useful to briefly discuss *supervised learning*, the most standard method to *train* these models. In the supervised learning framework, we are given data instances and an associated label for each data instance, i.e. (data instance, label) pairs. For example, the data instances might comprise of chest x-ray images, and the labels (one for each chest x-ray image) a binary yes/no to whether it shows the symptoms of pneumonia. *Training* the neural network model then consists of finding values for its parameters so that when it is fed in a data instance (chest x-ray) as input, it correctly outputs the corresponding label (yes/no on whether the chest x-ray has pneumonia.) To find these parameter values, we perform iterative *optimization* to guide the neural network parameters to appropriate values, using the given labels to provide supervision. Figure 2 shows a schematic of the supervised learning setup for deep learning.

Supervised learning is the most basic yet most critical method for training deep neural networks. As will be seen through the subsequent sections, there can be significant diversity in the kinds of (data, label) pairs used. Even in settings where clear (data, label) pairs are not possible to collect (Sections 6, 6.2), the training problem is often reformulated and recast into a supervised learning framework.

4.2 Multilayer Perceptrons

The first and most basic kind of deep neural network is the *multilayer perceptron.* These models consist of a stack of fully connected layers (matrix multiplications) interleaved with a nonlinear transform.

Despite their simplicity, they are useful for problems where the data might consist of a set of distinct, (possibly categorical) features, for example, tabular data. These models have more expressive power than logistic/linear regression, though those methods would be a good first step to try. One way to apply these models might be to first preprocess the data to compute the distinct set of features likely to be important, and use this as input. `https://github.com/rasbt/deeplearning-models` provides some implementations of some example multilayer perceptron architectures.

Scientific Examples One recent scientific example is given by the use of simple MLPs for pharamaceutical formulation [256], developing variants of a drug that is stable and safe for patient use.

4.3 Convolutional Neural Networks

These are arguably the most well known family of neural networks, and are very useful in working with any kind of image data. They are characterized by having convolutional layers, which allow the neural network to reuse parameters across different spatial locations of an image. This is a highly useful inductive bias for image data, and helping with efficiently learning good features, some, like Gabor filters, which correspond to traditional computer vision techniques. Convolutional neural networks (CNNs) have so many possible uses that we overview some of the most ubiquitous tasks separately below.

4.3.1 Image Classification

This is arguably the simplest and most well known application of convolutional neural networks. The model is given an input image, and wants to output a class — one of a (typically) mutually exclusive set of labels for that image. The earlier example, of mapping a chest x-ray image to a binary disease label, is precisely image classification.

Convolutional neural networks for image classification is an extremely common application of deep learning. There many different types of CNN models for classification: *VGG* — a simple stack of convolutional layers followed by a fully connected layer [214], *ResNets* — which are a family of convolutional networks of different sizes and depths and *skip connections* [79], *DenseNets* — another family of models where unlike standard neural networks, every layer in a "block" is connected to every other layer [94]. More recent, complex models include *ResNeXt* [253] and recently *EfficientNets*, which have separate scaling factors for network depth, width and the spatial resolution of the input image [223]. Tutorials, implementations and pretrained versions of many of these models can be found in the references given in Section 3.

Scientific Examples: Image classification has found many varied scientific applications, such as in analyzing cryoEM data [226] (with associated code `https://github.com/cramerlab/boxnet`). An especially large body of work has looked at *medical imaging* uses of image classification, specifically, using CNNs to predict disease labels. Examples range from ophthalmology [72], radiology (2D x-rays and 3D CT scans) [258, 5, 185],

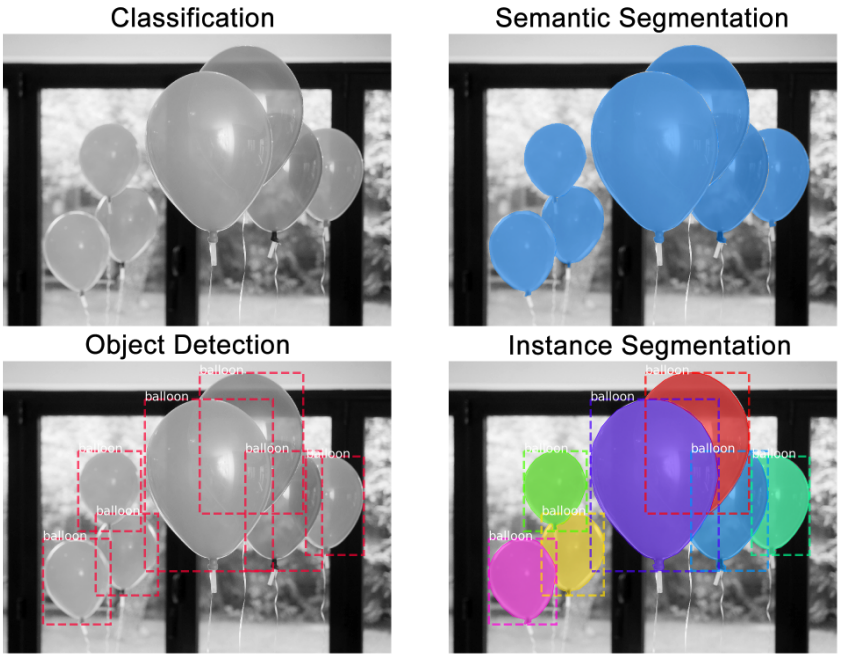

Figure 3: **Differences between Image Classification, Object Detection, Semantic Segmentation and Instance Segmentation tasks. Image source [1]** The figure illustrates the differences between classification, object detection, semantic segmentation and instance segmentation. In classification, the whole image gets a single label (balloons), while in object detection, each balloon is also localized with a bounding box. In semantic segmentation, all the pixels corresponding to balloon are identified, while in instance segmentation, each individual balloon is identified separately.

pathology [135, 55], analyzing brain scans (PET, fMRI) [202, 45]. An excellent survey of the numerous papers in this area is given by [228].

4.3.2 Object Detection

Image classification can be thought of as a global summary of the image. Object detection dives into some of the lower level details of the image, and looks at identifying and *localizing* different objects in the image. For example, given an input image of an outdoor scene having a dog, a person and a tree, object detection would look at both identifying the presence of the dog, person and tree and 'circle their location' in the image — specifically, put a *bounding box* around each of them. The supervised learning task is thus to take an input image and output the coordinates of these bounding boxes, as well as categorizing the kind of object they contain.

Like image classification, there are many high performing and well established convolutional architectures for object detection. Because of the intricacy of the output task, these models tend to be more complex with a *backbone* component (using an image classification model) and a *region proposal* component for bounding box proposals. But there are still many pretrained models available to download. One of the most successful early models was *Faster R-CNN* [192], which significantly sped up the slow bounding box proposal component. Since then there have been many improved models, including *YOLOv3* [191], and most recently *EfficientDets* [224]. Arguably the most popular recent architecture however has been *Mask R-CNN* and its variants [78, 248]. Mask R-CNN performs some segmentation as well as object detection (see below). Besides some of the resources mentioned in Section 3, a good source of code and models is `https://github.com/rbgirshick`, one of the key authors in a long line of these object

detection models. (Note though that there are many other popular implementations, such as https://github.com/matterport/Mask_RCNN.) This in depth article towardsdatascience object detection Faster R-CNN offers a detailed tutorial on downloading, setting up and training an object detection model, including helpful pointers to data collection and annotation (the latter using https://rectlabel.com/.) Most recently the Detectron2 system https://github.com/facebookresearch/detectron2 [248] builds on Mask R-CNN and offers many varied image task functionalities.

Scientific Examples: Object detection has also gained significant attention across different scientific applications. It has been used in many medical settings to localize features of interest, for example, tumor cells across different imaging modalities [125, 269] or fractures in radiology [199, 227].

4.3.3 Semantic Segmentation and Instance Segmentation

Segmentation dives into the lowest possible level of detail — categorizing every single image *pixel*. In semantic segmentation, we want to categorize pixels according to the high level group they belong to. For example, suppose we are given an image of a street, with a road, different vehicles, pedestrians, etc. We would like to determine if a pixel is part of any pedestrian, part of any vehicle or part of the road — i.e. label the image pixels as either pedestrian, vehicle or road. Instance segmentation is even more intricate, where not only do we want to categorize each pixel in this way, but do so separately for each instance (and provide instance specific bounding boxes like in object detection). The differences are illustrated in Figure 3 (sourced from [1].) Returning to the example of the image of the street, suppose the image has three pedestrians. In semantic segmentation, all of the pixels making up these three pedestrians would fall under the same category – pedestrian. In instance segmentation, these pixels would be further subdivided into those belonging to pedestrian one, pedestrian two or pedestrian three.

Because segmentation models must categorize every pixel, their output is not just a single class label, or a bounding box, but a full image. As a result, the neural network architectures for segmentation have a slightly different structure that helps them better preserve spatial information about the image. A highly popular and successful architecture, particularly for scientific applications, has been the *U-net* [196], which also has a 3d volumetric variant [33]. Other architectures include FCNs (Fully Convolutional Networks) [136], SegNet [9] and the more recent Object Contextual Representations [260]. A couple of nice surveys on semantic segmentation methods are given by towardsdatascience Semantic Segementation with Deep Learning and https://sergioskar.github.io/Semantic_Segmentation/.

For instance segmentation, Mask R-CNN [78] and its variants [248] have been extremely popular. This tutorial Mask R-CNN tutorial with code provides a step by step example application. The recent Detectron2 package [248] (https://github.com/facebookresearch/detectron2) also offers this functionality.

Scientific Examples: Out of all of the different types of imaging prediction problems, segmentation methods have been especially useful for (bio)medical applications. Examples include segmenting brain MR images [156, 236], identifying key regions of cells in different tissues [254, 217] and even studying bone structure [129].

4.3.4 Super-Resolution

Super resolution is a technique for transforming low resolution images to high resolution images. This problem has been tackled both using convolutional neural networks and supervised learning, as well as generative models.

Super resolution formally defined is an underdetermined problem, as there may be many possible high resolution mappings for a low resolution image. Traditional techniques imposed constraints such

as sparsity to find a solution. One of the first CNNs for super resolution, *SRCNN* [50] outlines the correspondences between sparse coding approaches and convolutional neural networks. More recently, *Residual Dense Networks* [270] have been a popular approach for super-resolution on standard benchmarks (with code available https://github.com/yulunzhang/RDN), as well as Predictive Filter Flow [114], (code: https://github.com/aimerykong/predictive-filter-flow) which has also looked at image denoising and deblurring. In some of the scientific applications below, *U-nets* have also been successful for super resolution.

Scientific Examples: Super resolution is arguably even more useful for scientific settings than standard natural image benchmarks. Two recent papers look at U-nets for super-resolution of fluorescence microscopy [245] (code: https://csbdeep.bioimagecomputing.com/) and electron microscopy [56]. Other examples include super resolution of chest CT scans [231] and Brain MRIs [31].

4.3.5 Image Registration

Image registration considers the problem of *aligning* two input images to each other. Particularly relevant to scientific applications, the two input images might be from different imaging modalities (e.g. a 3D scan and a 2D image), or mapping a moving image to a canonical template image (such as in MRIs.) The alignment enables better identification and analysis of features of interest.

The potential of image registration is primarily demonstrated through different scientific applications. At the heart of the technique is a convolutional neural network, often with an encoder-decoder structure (similar to the U-net [196]) to guide the alignment of two images. Note that while this underlying model is trained through supervised learning, many registration methods *do not require explicit labels*, using similarity functions and smoothness constraints to provide supervision. For example, [12] develop an unsupervised method to perform alignment for Brain MRIs. The code for this and several followup papers [13, 39] provides a helpful example for building off of and applying these methods https://github.com/voxelmorph/voxelmorph. Other useful resources include https://github.com/ankurhanda/gvnn (with corresponding paper [75]) a library for learning common parametric image transformations.

4.3.6 Pose Estimation

Pose estimation, and most popularly human pose estimation, studies the problem of predicting the pose of a human in a given image. In particular, a deep neural network model is trained to identify the location of the main joints, the *keypoints* (e.g. knees, elbows, head) of the person in the image. These predictions are combined with existing *body models* to get the full stick-figure-esque output summarizing the pose. (See Figure 4, sourced from [218], for an illustration.)

(2D) Human pose estimation is a core problem in computer vision with multiple benchmark datasets, and has seen numerous convolutional architectures developed to tackle it. Some of the earlier models include a multi-stage neural network introduced by [244], and a stacked hourglass model [158] that alternatingly combines high and low resolutions of the intermediate representations. More recently, HRNet [218], which keeps a high resolution representation throughout the model is a top performing architecture (code at https://github.com/leoxiaobin/deep-high-resolution-net.pytorch). Also of interest might be [24] provides an end-to-end system for multiperson pose detection in the corresponding code repository https://github.com/CMU-Perceptual-Computing-Lab/openpose.

Scientific Examples: Pose estimation has gained significant interest in neuroscience settings, where videos of animals are recorded, and automatically predicting poses in the image can help identify important behaviors. An example is given by [146, 147], with associated code http://www.mousemotorlab.org/deeplabcut.

Figure 4: **Pose Estimation. Image source [218]** The task of pose estimation, specifically multi-person 2D (human) pose-estimation is depicted in the figure. The neural network model predicts the positions of the main joints (keypoints), which are combined with a body model to get the stick-figure like approximations of pose overlaid on the multiple humans in the image. Variants of these techniques have been used to study animal behaviors in scientific settings.

4.3.7 Other Tasks with Convolutional Neural Networks

In the preceding sections, we have overviewed some of the most common tasks for which convolutional neural networks are used. However, there are many additional use cases of these models that we have not covered, including *video prediction* [57], *action recognition* [52] and *style transfer* [64]. We hope that the provided references and resources enable future investigation into some of these methods also.

4.4 Graph Neural Networks

Many datasets, such as (social) network data and chemical molecules have a *graph* structure to them, consisting of vertices connected by edges. An active area of research, graph neural networks, has looked at developing deep learning methods to work well with this kind of data. The input graph consists of nodes v having some associated feature vector h_v, and sometimes edges e_{uv} also having associated features $z_{e_{uv}}$. For example, nodes v might correspond to different atoms, and the edges e_{uv} to the different kinds of chemical bonds between atoms. At a high level, most graph neural networks compute useful information from the data by (i) using the feature vectors of the *neighbors* of each vertex v to compute information on the input graph instance (ii) using this information to update the feature vector of v. This process, which respects the connectivity of the graph, is often applied iteratively, with the final output either at the vertex level (Are meaningful vertex feature vectors computed?) or at the level of the full input graph (Is some global property of the entire graph correctly identified?)

Application Characteristics Problems where the data has an inherent graph structure, and the goal is to learn some function on this graph structure — either at the per vertex level or a global property of the entire graph. There are also spatio-temporal graph neural networks — performing predictions on graph

structures evolving over time.

Technical References Although most graph neural networks follow the high level structure of aggregating information from vertex neighbors and using this information to update feature vectors, there are many many different architectural variants, with connections to other neural network models such as convolutional nets and recurrent models. Recent work has also looked at spatio-temporal graph networks for problems like action recognition in video [124]. A nice unification of many of the first popular methods, such as [53, 15, 127], is given by [67]. A more recent survey paper [250], provides an *extremely comprehensive* overview of the different kinds of architectures, problems, benchmark datasets and open source resources. Some useful code repositories include https://github.com/rusty1s/pytorch_geometric, https://github.com/deepmind/graph_nets and https://github.com/dmlc/dgl, which together cover most of the popular deep learning frameworks.

Scientific Examples Graph neural networks have been very popular for several chemistry tasks, such as predicting molecular properties [53, 93, 67, 103], determining protein interfaces [60, 229] and even generating candidate molecules [41, 21]. A useful library for many of these chemistry tasks is https://github.com/deepchem, which also has an associated benchmark task [249]. A detailed tutorial of different graph neural networks and their use in molecule generation can be seen at https://www.youtube.com/watch?v=VXNjCAmb6Zw.

4.5 Neural Networks for Sequence Data

A very common attribute for data is to have a *sequential* structure. This might be frames in a video, amino acid sequences for a protein or words in a sentence. Developing neural network models to work with sequence data has been one of the most extensive areas of research in the past few years. A large fraction of this has been driven by progress on tasks in *natural language processing*, which focuses on getting computers to work with the language used by people to communicate. Two popular tasks in this area, which have seen significant advances, have been *machine translation* — developing deep learning models to translate from one language to another and *question answering* — taking as input a (short) piece of text and answering a question about it. In the following sections, we first overview some of the main NLP tasks that have driven forward sequence modelling and then the neural network models designed to solve these tasks.

4.5.1 Language Modelling (Next Token Prediction)

Language modelling is a training method where the deep learning model takes as input the tokens of the sequence up to time/position t, and then uses these to predict token $t+1$. This is in fact a *self-supervised* training method (see Section 6), where the data provides a natural set of labels without additional labelling needed. In the NLP context, the neural network is fed in a sequence of words, corresponding to a sentence or passage of text, and it tries to predict the next word. For example, given a sentence, "The cat sat on the roof", the network would first be given as input "The" and asked to predict "cat", then be fed in "The cat" and asked to predict "sat", and so on. (There are some additional details in implementation, but this is the high level idea.) Because of the easy availability of data/labels, and the ability to use language modelling at different levels — for words and even for characters, it has been a popular benchmark in natural language, and also for capturing sequence dependencies in scientific applications, such as protein function prediction [77, 80], and using the hidden representations as part of a larger pipeline for protein structure prediction in AlphaFold [205] (with opensourced code https://github.com/deepmind/deepmind-research/tree/master/alphafold_casp13.)

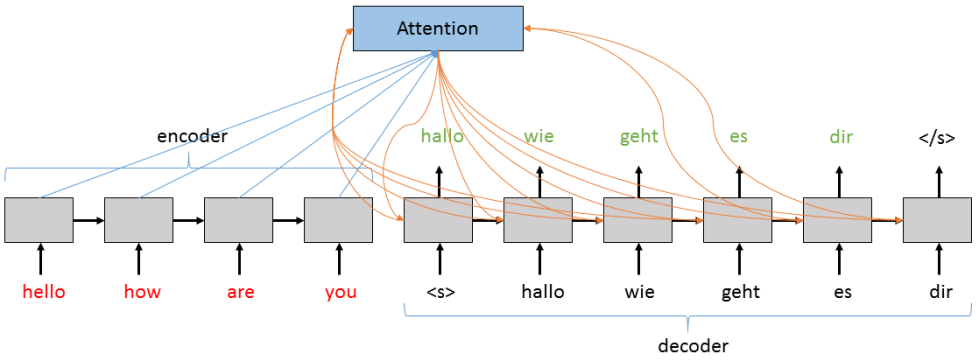

Figure 5: **Illustration of the Sequence to Sequence prediction task. Image source [267]** The figure shows an illustration of a Sequence to Sequence task, translating an input sentence (sequence of tokens) in English to an output sentence in German. Note the encoder-decoder structure of the underlying neural network, with the encoder taking in the input, and the decoder generating the output, informed by the encoder representations and the previously generated output tokens. In this figure, the input tokens are fed in one by one, and the output is also generated one at a time, which is the paradigm when using *Recurrent Neural Networks* as the underlying model. With *Transformer* models, which are now extremely popular for sequence to sequence tasks, the sequence is input all at once, significantly speeding up use.

4.5.2 Sequence to Sequence

Another very popular task for sequence data is *sequence to sequence* — transforming one sequence to another. This is precisely the setup for machine translation, where the model gets an input sentence (sequence) in say English, and must translate it to German, which forms the output sentence (sequence). Some of the first papers framing this task and tackling it in this way are [10, 221, 234]. Sequence to sequence tasks typically rely on neural network models that have an *encoder-decoder* structure, with the encoder neural network taking in the input sequence and learning to extract the important features, which is then used by the decoder neural network to produce the target output. Figure 5(sourced from [267]) shows an example of this. This paradigm has also found some scientific applications as varied as biology [23] and energy forcasting [145]. Sequence to sequence models critically rely on a technique called *attention*, which we overview below. For more details on this task, we recommend looking at some of the tutorials and course notes highlighted in Section 3.

4.5.3 Question Answering

One other popular benchmark for sequence data has been question answering. Here, a neural network model is given a paragraph of text (as context) and a specific question to answer on this context as input. It must then output the part of the paragraph that answers the question. Some of the standard benchmarks for this task are [83, 186], with `http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture10-QA.pdf` providing an excellent overview of the tasks and common methodologies. Question answering critically relies on the neural network model understanding the relevance and similarity of different sets of sequences (e.g. how relevant is this part of the context to the question of interest?). This general capability (with appropriate reformulation) has the potential to be broadly useful, both for determining similarity and relevance on other datasets, and for question answering in specialized domains [61].

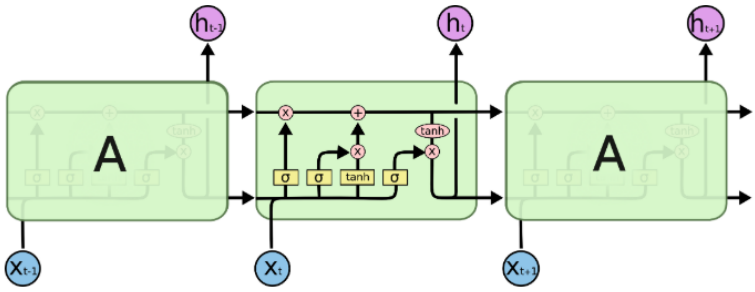

Figure 6: **Diagram of a Recurrent Neural Network model, specifically a LSTM (Long-Short Term Network). Image source [163]** The figure illustrates an LSTM network, a type of Recurrent Neural Network. We see that the input x_t at each timestep also inform the internal network state in the next timestep (hence a recurrent neural network) through a *gating mechanism*. This gating mechanism is called an LSTM, and consists of sigmoid and tanh functions, which transform and recombine the input for an updated internal state, and also emit an output. The mechanics of this gating process are shown in the middle cell of the figure.

4.5.4 Recurrent Neural Networks

Having seen some of the core tasks in deep learning for sequence data, these next few sections look at some of the key neural network models.

Recurrent neural networks (RNNs) were the first kind of deep learning model successfully used on many of the aforementioned tasks. Their distinguishing feature, compared to CNNs or MLPs (which are *feedforward* neural networks, mapping input straight to output), is that there are *feedback connections*, enabling e.g. the output at each timestep to become the input for the next timestep, and the preservation and modification of an internal state across timesteps. When RNNs are used for sequential data tasks, sequences are input token by token, with each token causing an update of the internal *cell state* of the RNN, and also making the RNN emit a token output. Note that this enables these models to work with *variable length* data — often a defining characteristic of sequence data. How the input is processed, cell state updated and output emitted are controlled by *gating functions* — see the technical references!

Application Characteristics: Problems where the data has a sequential nature (with different sequences of varying length), and prediction problems such as determining the next sequence token, transforming one sequence to another, or determining sequence similarities are important tasks.

Technical References: Research on sequence models and RNNs has evolved dramatically in just the past couple of years. The most successful and popular kind of RNN is a *bi-LSTM with Attention*, where LSTM (Long-Short Term Memory) [88] refers to the kind of gating function that controls updates in the network, bi refers to bidirectional (the neural network is run forwards *and* backwards on the sequence) and Attention is a very important technique that we overview separately below. (Some example papers [149, 150] and code resources `https://github.com/salesforce/awd-lstm-lm`.) This excellent post `https://colah.github.io/posts/2015-08-Understanding-LSTMs/` provides a great overview of RNNs and LSTMs in detail. (Figure 6 shows a diagram from the post revealing the details of the gating mechanisms in LSTMs.) The post also describes a small variant of LSTMs, Gated Recurrent Units (GRUs) which are also popular in practice [127]. While RNNs (really bi-LSTMs) have been very successful, they are often tricky to develop and train, due to their recursiveness presenting challenges with optimization (the vanishing/exploding gradients problem [87, 170, 76]), with performing fast model training (due to generating targets token by token), and challenges learning long term sequential dependencies. A new type of feedforward neural network architecture, the *Transformer* (overviewed below), was proposed to alleviate the first two of these challenges.

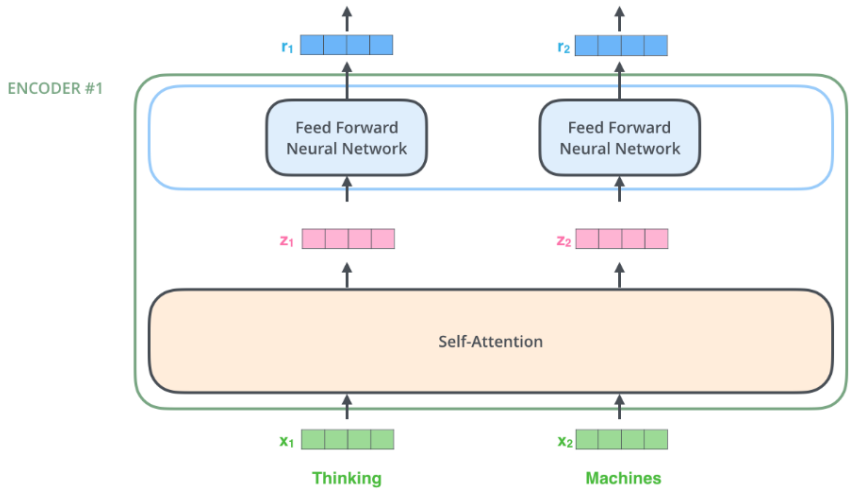

Figure 7: **Image of a couple of layers from a Transformer network. Image source [3]** The figure depicts the core sequence of layers that are fundamental to Transformer neural networks, a *self-attention* layer (sometimes called a self-attention head) followed by fully connected layers. Note that when working with sequence data, transformers take the entire input sequence all at once, along with positional information (in this case the input sequence being "Thinking Machines".)

Scientific Examples: RNNs have found several scientific applications for data with sequential structure, such as in genomics and proteomics [175, 132, 111].

4.5.5 Attention

A significant problem in using RNNs and working with sequential data is the difficulty in capturing *long range dependencies*. Long range dependencies are when tokens in the sequence that are very far apart from each other must be processed together to inform the correct output. RNNs process sequences in order, token by token, which means they must remember all of the important information from the earlier tokens until much later in the sequence — very challenging as the memory of these architectures is far from perfect. *Attention* [32, 11] is a very important technique that introduces *shortcut connections* to earlier tokens, which alleviates the necessity to remember important features for the duration of the entire sequence. Instead it provides a direct way to model long term dependencies — the neural network has the ability to *look back* and *attend* to what it deems relevant information (through learning) earlier in the input. A very nice overview of attention is provided by `https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html`. A variant of attention, *self-attention*, which can be used to help predictions on a single input sequence, is the core building block of Transformer models.

4.5.6 Transformers

While attention helped with challenges in long range dependencies, RNNs still remained slow to train and tricky to design (due to optimization challenges with vanishing/exploding gradients.) These challenges were inherent to their recurrent, token-by-token nature, prompting the proposal of a new *feedforward* neural network to work with sequential data, the Transformer [233], which critically relies on attentional mechanisms (the paper is in fact titled *Attention is All you Need*.) During training transformers take in the *entire* sequence as input all at once, but have *positional embeddings* that respects the sequential nature of the data. Transformers have been exceptionally popular, becoming the dominant approach to many natural language tasks and sequential tasks.

Application Characteristics: Problems where the data has a sequential nature and long range dependencies that need to be modelled. Given the large number of pretrained transformer models, they can also be very useful in settings where pretrained models on standard benchmarks can be quickly adapted to the target problem.

Technical References: The original transformer paper [233] provides a nice overview of the motivations and the neural network architecture. The model was designed with machine translation tasks in mind, and so consists of an *encoder* neural network and a *decoder* neural network. With transformers being adopted for tasks very different to machine translation, the encoder and decoder are often used in stand-alone fashions for different tasks — for example, the encoder alone is used for question answering, while the decoder is important for text generation. Two very accessible step by step tutorials on the transformer are The Annotated Transformer and The Illustrated Transformer. A nice example of some of the language modelling capabilities of this models is given by [180].

Since the development of the transformer, there has been considerable research looking at improving the training of these models, adjusting the self-attention mechanism and other variants. A very important result using the transformer has been BERT (Pretraining of deep Bi-directional Transformers for Language understanding) [43]. This paper demonstrates that performing *transfer learning* (see Section 5.1) using a transformer neural network can be extremely successful for many natural language tasks. (Some of the first papers showing the potential of transfer learning in this area were [92, 180], and since BERT, there have been followups which extend the model capabilities [257].) From a practical perspective, the development of transformers, BERT and transfer learning mean that there are many resources available online for getting hold of code and pretrained models. We refer to some of these in Section 3, but of particular note is `https://github.com/huggingface/transformers` which has an excellent library for transformer models. A good overview of BERT and transfer learning in NLP is given in `http://jalammar.github.io/illustrated-bert/`.

Scientific Examples: There have been several interesting examples of transformers used in scientific settings, such as training on protein sequences to find representations encoding meaningful biological properties [195], protein generation via language modelling [142], bioBERT [121] for text mining in biomedical data (with pretrained model and training code), embeddings of scientific text [18] (with code `https://github.com/allenai/scibert`) and medical question answering [237].

4.5.7 Other Tasks with Sequence Data

In the previous sections, we've given an overview of some of the important benchmark tasks for sequential data, and the types of deep learning models available to tackle them. As with convolutional networks, this is not a comprehensive overview, but hopefully thorough enough to help with generating ideas on possible applications and offering pointers to other useful related areas. A few other sequential data tasks that might be of interest are *structured prediction*, where the predicted output has some kind of structure, from tree structures (in e.g. parsing) [28, 246] to short, executable computer program structure [271] and *summarization*, where passages of text are summarized by a neural network [130, 273]. We'll also discuss *word embeddings* later in the survey.

4.6 Section Summary

In this section, we have overviewed supervised learning, some of the core neural network models and the kinds of important tasks they can be used for. As previously discussed, these topics span an extremely large area of research, so there are some areas, e.g. deep neural networks for set structured data [262, 113], modelling different invariances — invariances to specified Lie groups for application to molecular property prediction [58], spherical invariances [35, 36] not covered. But we hope the material and references presented help inspire novel contributions to these very exciting and rapidly evolving research directions.

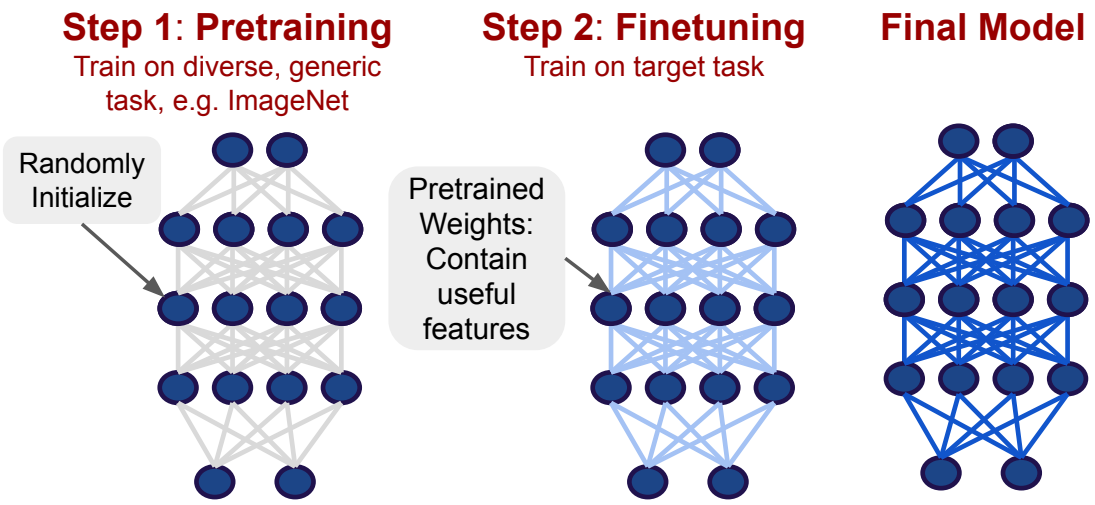

Figure 8: **The Transfer Learning process for deep neural networks.** Transfer learning is a two step process for training a deep neural network. Instead of intializing parameters randomly and directly training on the target task, we first perform a *pretraining* step, on some diverse, generic task. This results in the neural network parameters converging to a set of values, known as the *pretrained weights*. If the pretraining task is diverse enough, these pretrained weights will contain useful features that can be leveraged to learn the target task more efficiently. Starting from the pretrained weights, we then train the network on the target task, known as *finetuning*, giving us the final model.

5 Key (Supervised Learning) Methods

In the previous section we saw different kinds of neural network models, and the many different types of tasks they could be used for. To train the models for these tasks, we typically rely on the supervised learning methodology — optimize model parameters to correctly output given labels (the supervision) on a set of training data examples.

In more detail, the standard supervised learning method for deep neural networks consists of (i) collecting data instances (e.g. images) (ii) collecting *labels* for the data instances (e.g. is the image a cat or a dog) (iii) splitting the set of collected (data instance, label) into a training set, validation set and test set (iv) *randomly initializing* neural network parameters (iv) optimizing parameters so the network outputs the correct corresponding label given an input data instance on the training set (v) further tuning and validating on the validation and test sets.

In this section we overview methods that use variants of this process, for example initializing the neural network parameters differently or dealing with shifts between the training data and the test sets. In Section 6, we look at variants that reduce the dependence on collecting labels.

5.1 Transfer Learning

Through the preceding sections, we've made references to using *pretrained* models. This is in fact referring to a very important method for training deep neural networks, known as *transfer learning*. Transfer learning is a two step process for training a deep neural network model, a *pretraining* step, followed by a *finetuning* step, where the model in trained on the target task. More specifically, we take a neural network with parameters

randomly initialized, and first train it on a standard, generic task — the pretraining step. For example, in image based tasks, a common pretraining task is ImageNet [42], which is an *image classification* task on a large dataset of natural images. With an appropriate pretraining task that is generic and complex enough, the pretraining step allows the neural network to learn useful features, stored in its parameters, which can then be reused for the second step, *finetuning*. In finetuning, the pretrained neural network is further trained (with maybe some minor modifications to its output layer) on the *true target task* of interest. This process is illustrated in Figure 8. But being able to use the features it learned during pretraining often leads to boosts in performance and convergence speed of the target task, as well as needing less labelled data.

Because of these considerable benefits, transfer learning has been extraordinarily useful in many settings, particularly in computer vision [95], which had many early successful applications. As overviewed in Section 4.5.6, the recent development of models like ULMFiT [92] and especially BERT [43] has also made transfer learning extremely successful in natural language and sequential data settings, with recent work making the transfer learning process even more efficient [90, 201]. Most importantly, the ready availability of standard neural network architectures pretrained on standard benchmarks through many open sourced code repositories on GitHub (examples given in Section 3) has meant that downloading and finetuning a standard pretrained model has become the *de-facto standard* for most new deep learning applications.

Typically, performing transfer learning is an *excellent* way to start work on a new problem of interest. There is the benefit of using a well-tested, standard neural network architecture, aside from the knowledge reuse, stability and convergence boosts offered by pretrained weights. Note however that the precise effects of transfer learning are not yet fully understood, and an active research area [116, 184, 266, 159, 143, 181, 235] looks at investigating its exact properties. For transfer learning in vision [116, 266, 112] may be of particular interest for their large scale studies and pretraining recommendations.

5.2 Domain Adaptation

Related to transfer learning is the task of *domain adaptation*. In (unsupervised) domain adaptation, we have training data and labels in a *source* domain, but want to develop a deep learning model that will also work on a *target* domain, where the data instances may look different to those in the source domain, but the high level task is the same. For instance, our source domain many consist of images of handwritten digits (zero to nine) which we wish to classify as the correct number. But the target domain many have photographs of house numbers (from zero to nine), that we also wish to classify as the correct number. Domain adaptation techniques help build a model on the source domain that can also work (reasonably) well out-of-the-box on the shifted target domain.

The most dominant approach to domain adaptation in deep learning is to build a model that can (i) perform well on the source domain task, and (ii) learns features that are as invariant to the domain shift as possible. This is achieved through *jointly optimizing* for both of these goals. Returning to our example on handwritten digits and house number photographs, (i) corresponds to the standard supervised learning classification problem of doing well on the (source) task of identifying handwritten digits correctly while (ii) is more subtle, and typically involves explicitly optimizing for the *hidden layer representations* of handwritten digits and house number photographs to look the same as each other — domain invariance. Some popular ways to implement this include *gradient reversal* [62], minimizing a distance function on the hidden representations [137], and even *adversarial training* [63, 211]. More recently, [219] look at using *self-supervision* (see Section 6) to jointly train on the source and target domains, enabling better adaptation.

Other approaches to domain adaptation include translating data instances from the source to the target domain, and bootstrapping/co-training approaches (see Section 6.2). Some of these methods are overviewed in tutorials such as Deep Domain Adaptation in Computer Vision.

5.3 Multitask Learning

In many supervised learning applications, ranging from machine translation [2] to scientific settings [187, 176], neural networks are trained in a *multitask* way – predicting several different outputs for a single input. For example, in image classification, given an input medical image, we might train the network not only to predict a disease of interest, but patient age, history of other related disease, etc. This often has beneficial effects even if there is only one prediction of interest, as it provides the neural network with useful additional feedback that can guide it in learning the most important data features. (This can be so useful that sometimes auxiliary prediction targets are defined solely for this purpose.) Additionally, the prediction of multiple targets can mean that more data is available to train the model (only a subset of the data has the target labels of interest, but many more data instances have other auxiliary labels.) The most extreme version of this is to *simultaneously train* on two entirely different datasets. For example, instead of performing a pretraining/finetuing step, the model could be trained on both ImageNet and a medical imaging dataset at the same time.

Multitask learning is usually implemented in practice by giving the neural network multiple *heads*. The head of a neural network refers to its output layer, and a neural network with multiple heads has one head for each predictive task (e.g. one head for predicting age, one for predicting the disease of interest) but *shares* all of the other features and parameters, across these different predictive tasks. This is where the benefit of multitask learning comes from — the shared features, which comprise of most of the network, get many different sources of feedback. Implementing multitask learning often also requires careful choice of the way to weight the training objectives for these different tasks. A nice survey of some popular methods for multitask learning is given by https://ruder.io/multi-task/index.html#fn4, and a tutorial on some of the important considerations in http://hazyresearch.stanford.edu/multi-task-learning. One package for implementing multitask learning is found in https://github.com/SenWu/emmental and step-by-step example with code excerpts in towardsdatascience Multitask Learning: teach your AI more to make it better.

5.4 Weak Supervision (Distant Supervision)

Suppose it is very difficult to collect high quality labels for the target task of interest, and neither is there an existing, standard, related dataset and corresponding pretrained model to perform transfer learning from. How might one provide the deep learning model with enough supervision during the training process? While high quality labels might be hard to obtain, *noisy labels* might be relatively easy to collect. *Weak supervision* refers to the method of training a model on a dataset with these noisy labels (typically for future finetuning), where the noisy labels are often generated in an *automatic process*.

In computer vision (image based) tasks, some examples are: taking an image level label (for classification) and automatically inferring pixel level labels for segmentation [171], clustering hidden representations computed by a pretrained network as pseudo-labels [255], or taking Instagram tags as labels [143] for pretraining. In language tasks, examples are given by [153, 89, 264], which provide noisy supervision by assuming all sentences mentioning two entities of interest express a particular relation (also known as *distant supervision*). A nice overview of weak supervision and its connection to other areas is given in https://hazyresearch.github.io/snorkel/blog/ws_blog_post.html, with a related post looking specifically at medical and scientific applications http://hazyresearch.stanford.edu/ws4science.

5.5 Section Summary

In this section, we have overviewed some of the central supervised learning based methodologies for developing deep learning models. This is just a sampling of the broad collection of existing methods, and again, we hope that the descriptions and references will help facilitate further exploration of other approaches. One method not covered that might be of particular interest is *multimodal learning*, where neural networks are

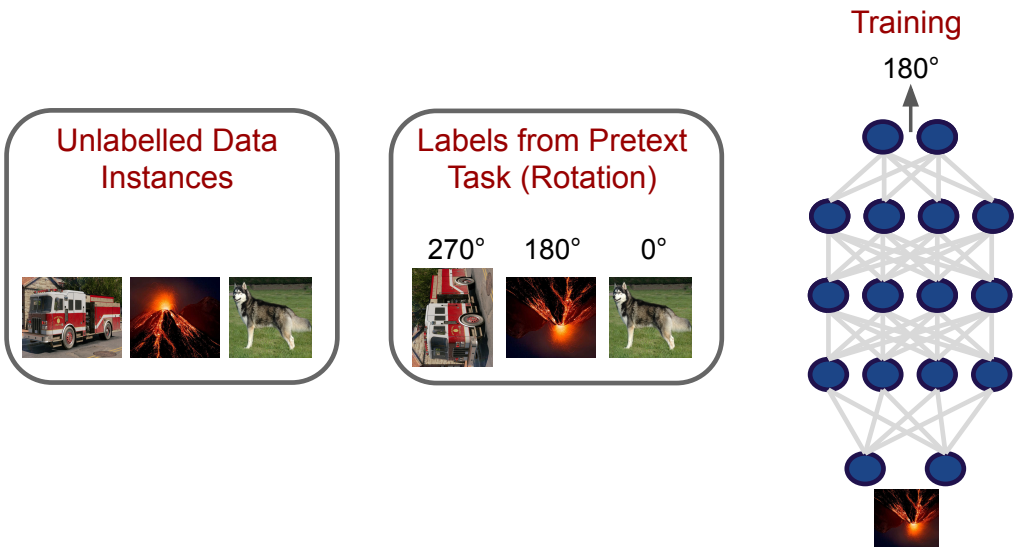

Figure 9: **Training neural networks with Self-Supervision.** The figure illustrates one example of a self-supervision setup. In self-supervision, we typically have a collection of unlabelled data instances, in this case images. We define a *pretext task*, that will automatically generate labels for the data instances. In this case, the pretext task is rotation — we randomly rotate the images by some amount and label them by the degree of rotation. During training, the neural network is given this rotated image and must predict the degree of rotation. Doing so also requires the neural network learn useful hidden representations of the image data in general, so after training with self-supervision, this neural network can then be successfully and efficiently finetuned on a downstream task.

simultaneously trained on data from different modalities, such as images and text [139, 238, 102]. Multimodal learning also provides a good example of the fact that it is often difficult to precisely categorize deep learning techniques as only being useful for a specific task or training regime. For example, we looked at language modelling for sequence tasks in this supervised learning section, but language modelling is also an example of self-supervision (Section 6) and generative models (Section 8.1). There are many rich combinations of the outlined methods in both this section and subsequent sections, which can prove very useful in the development of an end to end system.

6 Doing More with Less Data

Supervised learning methods, and specific variants such as transfer learning and multitask learning have been highly successful in training deep neural network models. However, a significant limitation to their use, and thus the use of deep learning, is the dependence on large amounts of *labelled* data. In many specialized domains, such as medicine, collecting a large number of high quality, reliable labels can be prohibitively expensive.

Luckily, in just the past few years, we've seen remarkable advances in methods that reduce this dependence, particularly *self-supervision* and *semi-supervised learning*. These approaches still follow the paradigm of training a neural network to map raw data instances to a specified label, but critically, these labels are not collected separately, but *automatically defined* via a pretext task. For example, we might take a dataset of images, rotate some of them, and then define the label as the degree of rotation, which is the prediction target for the neural network. This enables the use of *unlabelled data* in training the deep neural network. In

this section, we cover both self-supervision and semi-supervised learning as well as other methods such as data augmentation and denoising, all of which enable us to do more with less data.

6.1 Self-Supervised Learning

In self-supervision, a *pretext task* is defined such that labels can be *automatically* calculated directly from the raw data instances. For example, on images, we could rotate the image by some amount, label it by how much it was rotated, and train a neural network to predict the degree of rotation [66] — this setup is illustrated in Figure 9. This pretext task is defined without needing any labelling effort, but can be used to teach the network good representations. These representations can then be used as is or maybe with a little additional data for downstream problems. Arguably the biggest success of self-supervision has been *language modelling* for sequential data and specifically natural language problems, which we overviewed in Section 4.5.1. Below we outline some of the most popular and successful self-supervision examples for both image and sequential data. (A comprehensive list of self-supervision methods can also be found on this page https://github.com/jason718/awesome-self-supervised-learning.)

6.1.1 Self-Supervised Learning for Images

A recent, popular and simple self-supervised task for images is to predict image rotations [66]. Each image instance is transformed with one of four possible rotations and the deep learning model must classify the rotation correctly. Despite its simplicity, multiple studies have shown its success in learning good representations [266, 265, 112]. Another popular method examined in those studies is *exemplar* [51], which proposes a self-supervision task relying on invariance to *image transformations*. For example, we might take a source image of a cat, and perform a sequence of transformations, such as rotation, adjusting contrast, flipping the image horizontally, etc. We get multiple images of the cat by choosing many such sequences, and train the neural network to recognize these all as the same image.

Other methods look at using image patches as *context* to learn about the global image structure and important features. For example, [48] defines a pretext task where the relative locations of pairs of image patches must be determined, while [161] teaches a neural network to solve jigsaw puzzles. This latter task has been shown to be effective at large scales [69], with nice implementations and benchmarking provided by https://github.com/facebookresearch/fair_self_supervision_benchmark. A recent line of work has looked at using *mutual information* inspired metrics as a way to provide supervision on the relatedness of different image patches [84, 169, 7, 154], but these may be more intricate to implement. Many of these mutual information based metrics also rely on *contrastive losses* [30], which, at a high level, provides supervision to the network by making representations of a pair of similar inputs more similar than representations of a pair of different inputs. Very recently, a new self-supervision method, SimCLR [29], uses this to achieve high performance (one implementation at https://github.com/sthalles/SimCLR.)

Note that some of the *image registration* examples given in Section 4.3.5 are also examples of self-supervised learning, where some kind of domain specific similarity function can be automatically computed to assess the quality of the output. Such approaches may be relevant to other domains, and are useful to explore. A great set of open-sourced implementations of many of self-supervision methods is provided by https://github.com/google/revisiting-self-supervised.

6.1.2 Self-Supervised Learning for Sequential (Natural Language) Data

While research on self-supervision techniques for images has been extremely active, the strongest successes of this framework have arguably been with sequential data, particularly text and natural language. The sequential structure immediately gives rise to effective self-supervision pretext tasks. Two dominant classes of pretext tasks operate by either (i) using neighboring tokens of the sequence as input *context* for predicting a

target token (ii) taking in all tokens up to a particular position and predicting the next token. The latter of these is *language modelling*, which was overviewed in Section 4.5.1. The former is the principle behind *word embeddings*.

Word embeddings have been critical to solving many natural language problems. Before the recent successes of full fledged transfer learning in language (Section 5.1) this simple self-supervised paradigm was where knowledge reuse was concentrated, and formed a highly important component of any deep learning system for natural language (sequential) data. From a scientific perspective, learning word embeddings for sequential data has the potential to identify previously unknown similarities in the data instances. It has already found interesting uses in aiding with the automatic analysis of scientific texts, such as drug name recognition systems [131], biomedical named entity recognition [73], identifying important concepts in materials science [230] and even detecting chemical-protein interactions [37].

The key fundamental ideas of word embeddings are captured in the *word2vec* framework [152, 151], the original framework relying on either a *Continuous-Bag-of-Words* (CBOW) neural network or a *Skip-Gram* neural network. Actually, both of these models are less neural networks and more two simple matrix multiplications, with the first matrix acting as a *projection*, and giving the desired embedding. In CBOW, the context — defined as the neighborhood words — are input, and the model must correctly identify the target output word. In Skip-Gram, this is reversed, with the center word being input, and the context being predicted. For example, given a sentence "There is a cat on the roof", with the target word being cat, CBOW would take in the vector representations of (There, is, a, on, the, roof) and output "cat", while Skip-Gram would roughly swap the inputs and outputs. The simplicity of these methods may make them more suitable for many tasks compared to language modelling. Two nice overviews of the these methods are given by Introduction to Word Embeddings and word2vec, and `https://ruder.io/word-embeddings-1/`. Other embedding methods include [173, 123].

6.1.3 Self-Supervision Summary

In this section we have outlined many of the interesting developments in self-supervised learning, a very successful way to make use of unlabelled data to learn meaningful representations, either for analysis or other downstream tasks. Self-supervision can be effectively used along with other techniques. For example, in the language modelling application, we saw it used for transfer learning (Section 5.1), where a deep learning model is first pretrained using the language modelling self supervision objective, and then finetuned on the target task of interest. In the following section, we will other ways of combining self-supervision with labelled data.

6.2 Semi-Supervised Learning

While collecting *large* labelled datasets can be prohibitively expensive, it is often possible to collect a smaller amount of labelled data. When assembling a brand new dataset, a typical situation is having a small amount of labelled data and a (sometimes significantly) larger number of data instances with no labels. *Semi-supervised learning* looks at precisely this setting, proposing techniques that enable effective learning on labelled and unlabelled data. Below we overview some of the popular methods for semi-supervised learning.

6.2.1 Self-Supervision with Semi-Supervised Learning

Following on from the previous section, one natural way to make use of the unlabelled data is to use a self-supervised pretext task. To combine this with the labelled data, we can design a neural network that has *two different outputs heads* (exactly as in multitask learning, see Section 5.3), with one output head being used for the labelled data, and the other for the self-supervised objective on the unlabelled data. Importantly,

this means that the features learned by the neural network are *shared* between the labelled *and* unlabelled data, leading to better representations. This simple approach has been shown to be very effective [266, 265].

6.2.2 Self-Training (Bootstrapping)

Self-training, sometimes also referred to as *bootstrapping* or *pseudo-labels*, is an iterative method where a deep neural network is first developed in a supervised fashion on the labelled data. This neural network is then used to provide (pseudo) labels to the unlabelled data, which can then be used in conjunction with the labelled data to train a new, more accurate neural network. This approach often works well and can even be repeated to get further improvements. There are a couple of common details in implementation — often when adding the neural network pseudo-labelled data, we only keep the most *confidently* pseudo-labelled examples. These pseudo-labelled examples may also be used for training with a different objective function compared to the labelled data. One of the early papers proposing this method was [120], with a more recent paper [252] demonstrating significant successes at large scale. Other variants, including *mean teacher* [225], *temporal ensembling* [119] and the recent *MixMatch* [19] also primarily use the self-training approach, but incorporate elements of *consistency* (see below). There are nice open sourced implementations of these methods, such as `https://github.com/CuriousAI/mean-teacher` for mean teacher and `https://github.com/google-research/mixmatch` and `https://github.com/YU1ut/MixMatch-pytorch` for MixMatch.

6.2.3 Enforcing Consistency (Smoothness)

An important theme in many semi-supervised methods has been to provide supervision on the unlabelled data through enforcing *consistency*. If a human was given two images A and B, where B was a slightly perturbed version of A (maybe blurred, maybe some pixels obscured or blacked out), they would give these images the same label — consistency. We can also apply this principle to provide feedback to our neural network on the unlabelled data, combining it with the labelled data predictions as in multitask learning (Section 5.3) to form a semi-supervised learning algorithm. A popular method on enforcing consistency is *virtual adversarial training* [155], which enforces consistency across carefully chosen image perturbations. Another paper, *unsupervised data augmentation* [251], uses standard data augmentation techniques such as cutout [44] for images and back translation for text [206] to perturb images and enforces consistency across them. [265] uses consistency constraints along with other semi-supervised and self-supervised techniques in its full algorithm.

6.2.4 Co-training

Another way to provide feedback on unlabelled data is to train two (many) neural network models, each on a different *view* of the raw data. For example, with text data, each model might see a different part of the input sentence. These models can then be given feedback to be maximally consistent with each other, or with a different model which sees all of the data, or even used for self-training, with each different model providing pseudo labels on the instances it is most confident on. This post `https://ruder.io/semi-supervised/` gives a nice overview of different co-training schemes, and [34, 179, 74] are some recent papers implementing this in text and images.

6.2.5 Semi-Supervised Learning Summary

Semi-supervised learning is a powerful way to reduce the need for labelled data and can significantly boost the efficacy of deep learning models. Semi-supervised learning can be applied in any situation where a meaningful task can be created on the unlabelled data. In this section we have overviewed some natural ways to define such tasks, but there may be many creative alternatives depending on the domain of interest. We hope the references will provide a helpful starting point for implementation and further exploration!

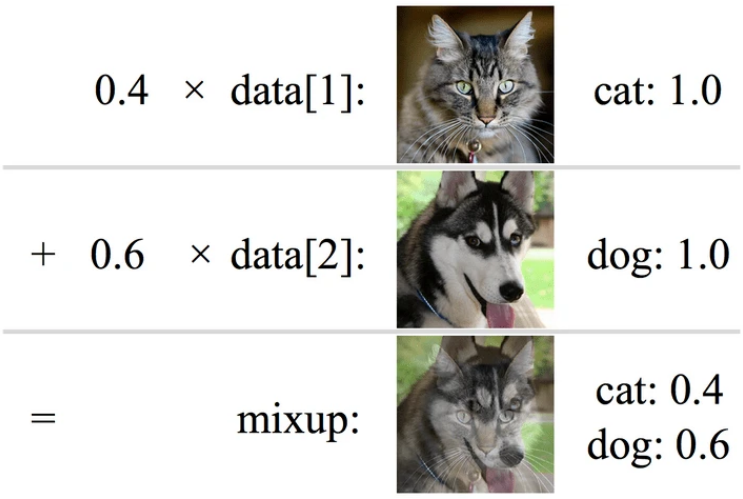

Figure 10: **An illustration of the Mixup data augmentation technique. Image source [40]** The figure provides an example of the Mixup data augmentation method — an image of a cat and an image of a dog are linearly combined, with 0.4 weight on the cat and 0.6 weight on the dog, to give a new input image shown in the bottom with a smoothed label of 0.4 weight on cat and 0.6 weight on dog. Mixup has been a very popular and successful data augmentation method for image tasks.

6.3 Data Augmentation

As depicted in Figure 1, *data augmentation* is an important part of the deep learning workflow. Data augmentation refers to the process of artificially increasing the size and diversity of the training data by applying a variety of transformations to the raw data instances. For example, if the raw instances were to consist of images, we might artificially *pad* out the image borders and then perform an off center (random) *crop* to give us the final augmented image instance. Aside from increasing the size and diversity of the data, data augmentation offers the additional benefit of encouraging the neural network to be robust to certain kinds of common transformations of data instances. In this section, we overview some of the most popular data augmentation techniques for image and sequential data. These techniques will typically already be part of many open sourced deep learning pipelines, or easy to invoke in any mainstream deep learning software package. There are also some specific libraries written for augmentations, for example *imgaug* https://github.com/aleju/imgaug, *nlpaug* https://github.com/makcedward/nlpaug and *albumentations* https://github.com/albumentations-team/albumentations.

6.3.1 Data Augmentation for Image Data

Simple augmentations for image data consider transformations such as *horizontal flips* or *random crops* (padding the image borders and taking an off center crop.) Inspired by these simple methods are two very successful image augmentation strategies, *cutout* [44], which removes a patch from the input image, and RICAP [222], which combines patches from four different input image to create a new image (with new label a combination of the original labels.) This somewhat surprising latter technique of combining images has in fact shown to be very successful in *mixup* [268], another data augmentation strategy where linear combinations of images (instead of patches) are used. (This strategy has also been combined with cutout in the recently proposed *cutmix* augmentation strategy [261], with code https://github.com/clovaai/CutMix-PyTorch.)

Other useful augmentation strategies include *TANDA* [188] which learns a model to compose data augmentations, the related *randaugment* [38], choosing a random subset of different possible augmentations, population based augmentation [85] which randomly searches over different augmentation policies, [91]

applying color distortions to the image and the recently proposed *augmix* [82] (code https://github.com/google-research/augmix.)

6.3.2 Data Augmentation for Sequence Data

Data augmentation for sequential data typically falls into either (i) directly modifying the input sequence, or (ii) in the case of *sequence to sequence* tasks (Section 4.5.2), increasing the number of input-output sequences through noisy translation with the neural network. When directly modifying the input sequence, common perturbations include randomly deleting a sequence token (comparable to the masking approach used in [43]), swapping sets of sequence tokens, and replacing a token with its *synonym*. This latter strategy is usually guided by word embeddings [239] or contextualized word embeddings [110]. Examples of combining these transformations are given by [243, 98], with code repositories such as https://github.com/makcedward/nlpaug providing some simple implementations.

The other dominant approach to data augmentation of sequences is using sequence-to-sequence models to generate new data instances, known as *back-translation* [206, 54]. Concretely, suppose we have a model to translate from English sequences to German sequences. We can take the output German sequence and use existing tools/noisy heuristics to translate it back to English. This gives us an additional English-German sequence pair.

6.4 Data (Image) Denoising

When measuring and collecting high dimensional data, noise can easily be introduced to the raw instances, be they images or single-cell data. As a result there has been significant interest and development of deep learning techniques to denoise the data. Many of these recent methods work even without paired noisy and clean data samples, and many be applicable in a broad range of settings. For instance, *Noise2Noise* [122] uses a *U-net* neural network architecture to denoise images given multiple noisy copies. The recent *Noise2Self* [14] (with code: https://github.com/czbiohub/noise2self) frames denoising as a self-supervision problem, using different subsets of features (with assumed independent noise properties) to perform denoising, applying it to both images as well as other high dimensional data.

7 Interpretability, Model Inspection and Representation Analysis

Many standard applications of deep learning (and machine learning more broadly) focus on *prediction* — learning to output specific target values given an input. Scientific applications, on the other hand, are often focused on *understanding* — identifying underlying mechanisms giving rise to observed patterns in the data. When applying deep learning in scientific settings, we can use these observed phenomena as prediction targets, but the ultimate goal remains to understand what attributes give rise to these observations. For example, the core scientific question might be on how certain amino acid sequences (encoding a protein) give rise to particular kinds of protein function. While we might frame this as a prediction problem, training a deep neural network to take as input an amino acid sequence and output the predicted properties of the protein, we would ideally like to understand how that amino acid sequence resulted in the observed protein function.

To answer these kinds of questions, we can turn to *interpretability* techniques. Interpretability methods are sometimes equated to a fully understandable, step-by-step explanation of the model's decision process. Such detailed insights can often be intractable, especially for complex deep neural network models. Instead, research in interpretability focuses on a much broader suite of techniques that can provide insights ranging from (rough) *feature attributions* — determining what input features matter the most, to *model inspection* — determining what causes certain neurons in the network to fire. In fact, these two examples also provide a rough split in the type of interpretability method.

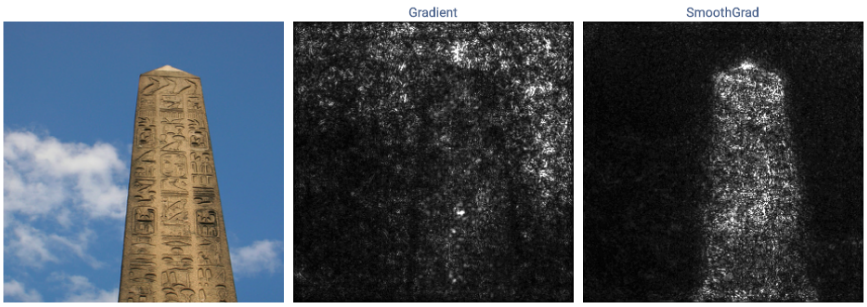

Figure 11: **The output of SmoothGrad, a type of saliency map. Image source [215]** The figure shows the original input image (left), raw gradients (middle), which are often too noisy for reliable feature attributions, and SmoothGrad (right), a type of saliency map that averages over perturbations to produce a more coherent feature attribution visualization the input. In particular, we can clearly see that the monument in the picture is important for the model output.

One large set of methods (which we refer to as Feature Attribution and Per Example Interpretability) concentrates on taking a specific input along with a trained deep neural network, and determining what features of the input are most important. The other broad class of techniques looks at taking a trained model, and a *set* of inputs, to determine what different parts of the network have learned (referred to as Model Inspection and Representational Analysis). This latter set of methods can be very useful in revealing important, hidden patterns in the data that the model has implicitly learned through being trained on the predictive task. For example, in [118], which looks at machine translation, representation analysis techniques are used to illustrate latent linguistic structure learned by the model. We overview both sets of methods below.

7.1 Feature Attribution and Per Example Interpretability

We start off by overviewing some of the popular techniques used to provide *feature attribution* at a per example level, answering questions such as which parts of an input image are most important for a particular model prediction. These techniques can be further subcategorized as follows:

7.1.1 Saliency Maps and Input Masks

At a high level, saliency maps take the gradient of the *output prediction* with respect to the *input*. This gives a *mask* over the input, highlighting which regions have large gradients (most important for the prediction) and which have smaller gradients. First introduced by [213], there are many variants of saliency maps, such as Grad-CAM [204], SmoothGrad [215], IntGrad [220], which make the resulting feature attributions more robust. These and other methods are implemented in https://github.com/PAIR-code/saliency. Note that while these methods can be extremely useful, their predictions are not perfect [105], and must be validated further.

Closely related to these saliency methods is [166], which provides the ability to inspect the kinds of features causing neurons across different hidden layers to fire. The full, interactive paper can be read at https://distill.pub/2018/building-blocks/ with code and tutorials available at https://github.com/tensorflow/lucid.

Many other techniques look at computing some kind of input mask, several of them using *deconvolutional* layers, first proposed by [263] and built on by [106] and [20]. Other work looks at directly optimizing to find a sparse mask that will highlight the most important input features [59] (with associated code https://github.com/jacobgil/pytorch-explain-black-box) or finding such a mask through an iterative

Figure 12: **Visualization of the kinds of features hidden neurons have learned to detect. Image source [165]** This figure, from [165], illustrates the result of optimizing inputs to show what features hidden neurons have learned to recognize. In this example, the hidden neuron has learned to detect (especially) soccer balls, tennis balls, baseballs, and even the legs of soccer players.

algorithm [25].

7.1.2 Feature Ablations and Perturbations

Related to some of masking approaches above, but with enough differences to categorize separately are several methods that isolate the crucial features of the input either by performing feature *ablations* or computing perturbations of the input and using these perturbations along with the original input to inform the importance of different features.

Arguably the most well known of the ablation based approaches is the notion of a *Shapely value*, first introduced in [207]. This estimates the importance of a particular feature x_0 in the input by computing the predictive power of a subset of input features containing x_0 and averaging over all possible such subsets. While Shapely values may be expensive to compute naively for deep learning, follow on work [140] has proposed more efficient (and expressive) variants, with highly popular opensourced implementation: `https://github.com/slundberg/shap`.

The *shap* opensourced implementation above also unifies some related approaches that use *perturbations* to estimate feature values. One such approach is LIME [194], which uses multiple local perturbations to enable learning an interpretable local model. Another is DeepLIFT, which uses a reference input to compare activation differences [210], and yet another approach, Layer-wise Relevance Propagation [6] looks at computing relevance scores in a layerwise manner.

Other work performing ablations to estimate feature importance includes [275] (with code `https://github.com/lmzintgraf/DeepVis-PredDiff`), while [59], described in Section 7.1.1 has elements of using input perturbations.

7.2 Model Inspection and Representational Analysis

In this second class of interpretability methods, the focus is on gaining insights not at a *single* input example level, but using a set of examples (sometimes implicitly through the trained network) to understand the salient properties of the data. We overview some different approaches below.

7.2.1 Probing and Activating Hidden Neurons

A large class of interpretability methods looks at either (i) *probing* hidden neurons in the neural network — understanding what kinds of inputs it activates for (ii) directly optimizing the *input* to activate a hidden neuron. Both of these techniques can provide useful insights into what the neural network has chosen to pay attention to, which in turn corresponds to important properties of the data.

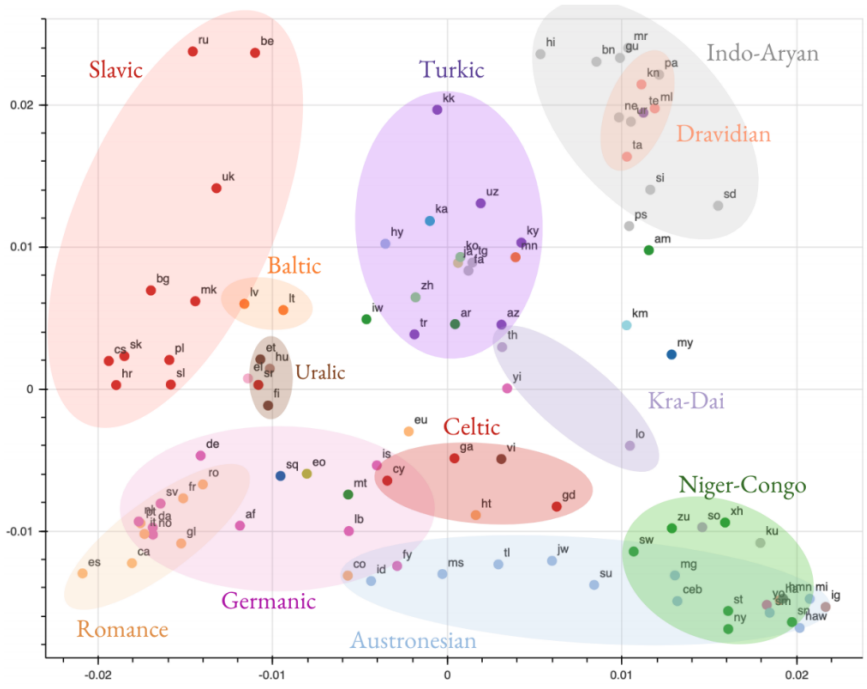

Figure 13: **Clustering neural network hidden representations to reveal linguistic structures. Image source [118]** In work on analyzing multilingual translation systems [118], representational analysis techniques are used to compute similarity of neural network (Transformer) hidden representations across different languages. Performing clustering on the result reveals grouping of different language representations (each language a point on the plot) according to language families, which affect linguistic structure. Importantly, this analysis uses the neural network to identify key properties of the underlying data, a mode of investigation that might be very useful in scientific domains.

Several papers falls into the probing category [259, 272], with an especially thorough study given by *Network Dissection* [17]. Here here hidden neurons are categorized by the kinds of features they respond to. The paper website `http://netdissect.csail.mit.edu/` contains method details as well as links to the code and data.

The other broad category of methods take a neural network, fix its parameters, and optimize the *input* to find the kinds of features that makes some hidden neuron activate. There are several papers using this approach, but of particular note is *Feature Visualization* [165], with an interactive article and code at: `https://distill.pub/2017/feature-visualization/`. Followup work, *Activation Atlases* [26] (with page `https://distill.pub/2019/activation-atlas/`), does this across many different concepts, providing a full mapping of the features learned by the neural network. More recently [164] has used this as a building block to further understand how certain computations are performed in a neural network. Also related is [104], which looks at finding linear combinations of hidden neurons that correspond to interpretable concepts.

7.2.2 Dimensionality Reduction on Neural Network Hidden Representations

In many standard scientific settings, e.g. analyzing single cell data, dimensionality reduction methods such as PCA, t-SNE [141], UMAP [148] are very useful in revealing important factors of variation and critical differences in the data subpopulations e.g. tumor cells vs healthy cells. Such methods can also be used on the *hidden activations* (over some input dataset) of a neural network. Through the process of being trained on some predictive task, the neural network may implicitly learn these important data attributes in its hidden representations, which can then be extracted through dimensionality reduction methods.

7.2.3 Representational Comparisons and Similarity

Related to more standard approaches of dimensionality reduction and clustering, a line of work has studied *comparing* hidden representations across different neural network models. Early work applied matching algorithms [126] with follow on approaches using *canonical correlation analysis* [183, 157] (with associated code https://github.com/google/svcca.) This latter approach has been used to identify and understand many representational properties in natural language applications [118, 16, 235] and even in modelling the mouse visual cortex as an artificial neural network [208]. Another recent technique uses a kernel based approach to perform similarity comparisons [115] (with code https://colab.sandbox.google.com/github/google-research/google-research/blob/master/representation_similarity/Demo.ipynb.)

7.3 Technical References

The preceding sections contain many useful pointers to techniques and associated open sourced code references. One additional reference of general interest may be https://christophm.github.io/interpretable-ml-book/ a fully open sourced book on interpretable machine learning. This focuses slightly more on more traditional interpretability methods, but has useful overlap with some of the techniques presented above and may suggest promising open directions.

8 Advanced Deep Learning Methods

The methods and tasks overviewed in the survey so far — supervised learning, fundamental neural network architectures (and their many different tasks), different paradigms like transfer learning as well as ways to reduce labelled data dependence such as self-supervision and semi-supervised learning — are an excellent set of first approaches for any problem amenable to deep learning. In most such problems, these approaches will also suffice in finding a good solution.

Occasionally however, it might be useful to turn to more advanced methods in deep learning, specifically *generative models* and *reinforcement learning*. We term these methods advanced as they are often more intricate to implement, and may require specific properties of the problem to be useful, for example an excellent environment model/simulator for reinforcement learning. We provide a brief overview of these methods below.

8.1 Generative Models

At a high level, generative modelling has two fundamental goals. Firstly, it seeks to *model* and enable *sampling* from high dimensional data distributions, such as natural images. Secondly, it looks to learn low(er) dimensional latent encodings of the data that capture key properties of interest.

To achieve the first goal, generative models take samples of the high dimensional distribution as input, for example, images of human faces, and learn some task directly on these data instances (e.g. encoding and then decoding the instance or learning to generate synthetic instances indistinguishable from the given data samples or generating values per-pixel using neighboring pixels as context). If generative modelling achieved *perfect* success at this first goal, it would make it possible to continuously sample 'free' data instances! Such perfect success is extremely challenging, but the past few years has seen enormous progress in the diversity and fidelity of samples from the data distribution.

For the second goal, learning latent encodings of the data with different encoding dimensions correspond to meaningful factors of variation, having an explicit encoder-decoder structure in the model can be helpful in encouraging learning such representations. This is the default structure for certain kinds of generative models

Figure 14: **Human faces generated from scratch by StyleGAN2. Image source [100]** The figure shows multiple human face samples from StyleGAN2 [100]. While perfectly modelling and capture full diversity of complex data distributions like human faces remains challenging, the quality and fidelity of samples from recent generative models is very high.

such as *variational autoencoders* [109] but has also been adopted into other models, such as BigBiGAN [49], a type of *generative adversarial network*. In the following sections we overview some of these main types of generative models.

8.1.1 Generative Adversarial Networks

Arguably the most well known of all different types of generative models, Generative Adversarial Networks, commonly known as GANs, consist of two neural networks, a *generator* and a *discriminator*, which are pitted in a game against each other. The generator takes as input a *random noise vector* and tries to output samples that look like the data distribution (e.g. synthesize images of peoples faces), while the discriminator tries to distinguish between true samples of the data, and those synthesized by the generator. First proposed in [68], GANs have been an exceptionally popular research area, with the most recent variations, such as BigGAN [22] (code: `https://github.com/ajbrock/BigGAN-PyTorch`), BigBiGAN [49] and StyleGAN(2) [100] (code: `https://github.com/NVlabs/stylegan2`) able to generate incredibly realistic images.

Unconditional GANs vs Conditional GANs The examples given above are all *unconditional* GANs, where the data is generated with only a random noise vector as input. A popular and highly useful variant are *conditional* GANs, where generation is conditioned on additional information, such as a label, or a 'source' image, which might be *translated* to a different style. Examples include *pix2pix* [96] (code: `https://phillipi.github.io/pix2pix/`), cycleGAN [274], and applications of these to videos [27].

GANs have found many scientific applications, from performing data augmentation in medical image settings [65] to protein generation [193]. The 'adversarial' loss objective of GANs can make them somewhat tricky to train, and useful implementation advice is given in `https://www.fast.ai/2019/05/03/decrappify/`, and (for conditional GANs) is included in `https://github.com/jantic/DeOldify`.

8.1.2 Variational Autoencoders

Another type of generative model is given by the *variational autoencoder*, first proposed by [109]. VAEs have an encoder decoder structure, and thus an explicit latent encoding, which can capture useful properties of the data distribution. They also enable estimation of the *likelihood* of a sampled datapoint — the probability of its occurrence in the data distribution. VAEs have also been extremely popular, with many variations and extensions proposed [216, 99, 107, 71]. Because of the explicit latent encoding and the ability to estimate likelihoods, they have also found use cases in various scientific settings, such as for modelling gene expression in single-cell RNA sequencing [138].

8.1.3 Autoregressive Models

Yet another type of generative model is *autoregressive models*, which take in inputs sequentially and use those to generate an appropriate output. For instance, such models may take in a sequence of pixel values (some of them generated at a previous timestep) and use these to generate a new pixel value for a specific spatial location. Autoregressive models such as PixelRNN [168], PixelCNN (and variants) [232, 200] and the recently proposed VQ-VAE(2) [189] (code: https://github.com/rosinality/vq-vae-2-pytorch) offer very high generation quality.

8.1.4 Flow Models

A relatively new class of generative models, *flow models*, looks at performing generation using a sequence of invertible transformations, which enables the computation of *exact likelihoods*. First proposed in [46, 47], performing an expressive but tractable sequence of invertible transformations is an active area of research [108, 86]. A nice introduction to normalizing flows is given in this short video tutorial https://www.youtube.com/watch?v=i7LjDvsLWCg&feature=youtu.be.

8.2 Reinforcement Learning

Reinforcement learning has quite a different framing to the techniques and methods introduced so far, aiming to solve the *sequential decision making* problem. It is typically introduced with the notions of an *environment* and an *agent*. The agent can take a *sequence of actions* in the environment, each of which affect the environment *state* in some way, and also result in possible *rewards* (feedback) — 'positive' for good sequences of actions resulting in a 'good' state and 'negative' for bad sequences of actions leading to a 'bad' state. For example, in a game like chess, the state is the current position of all pieces in play (the game state), an action the moving of a piece, with a good sequence of actions resulting in a win, a bad sequence of actions in a loss and the reward might be one or zero depending on having a win or loss respectively.

With this being the setup, the goal of reinforcement learning is to learn, through interaction with the environment, good sequences of actions (typically referred to as a *policy*). Unlike supervised learning, feedback (the reward) is typically given only after performing the *entire sequence* of actions. Specifically, feedback is *sparse* and *time delayed*. There are a variety of different reinforcement learning use cases depending on the specifics of the problem.

8.2.1 RL with an Environment Model/Simulator

Some of the most striking results with RL, such as AlphaGoZero [212], critically use an environment model/simulator. In such a setting, a variety of learning algorithms [242, 203, 128] (some code: https://github.com/openai/baselines) can help the agent learn a good sequence of actions, often through simultaneously learning a *value function* — a function that determines whether a particular *environment state* is beneficial or not. Because the benefit of an environment state may depend on the entire sequence of actions (some still in the future), RL is very important in properly assessing the value of the environment state, through implicitly accounting for possible future actions. Combining value functions with traditional search algorithms has been a very powerful way to use RL, and may be broadly applicable to many domains.

Specifically, if developing a solution to the problem is multistep in nature, with even a noisy validation possible in simulation, using RL to learn a good value function and combining that with search algorithms may lead to discovering new and more effective parts of the search space. Approaches like these have gained traction in considering RL applications to fundamental problems in both computer systems, with [144] providing a survey and a new benchmark, and machine learning systems [174], in designing task-specific neural network models. The latter has recently also resulted in scientific use cases — designing neural

networks to emulate complex processes across astronomy, chemistry, physics, climate modelling and others [101].

8.2.2 RL without Simulators

In other settings, we don't have access to an environment model/simulator, and may simply have records of sequences of actions (and the ensuing states and rewards). This is the *offline* setting. In this case, we may still try to teach an agent a good policy, using the observed sequences of actions/states/rewards in conjunction with *off-policy* methods [209, 182, 134], but thorough validation and evaluation can be challenging. Evaluation in off-policy settings often uses a statistical technique known as *off-policy policy evaluation* (example algorithms include [178, 133]). In robotics, reinforcement learning literature has looked at performing transfer learning between policies learned in simulation and policies learned on real data [198]. A thorough overview of deep reinforcement learning is given in http://rail.eecs.berkeley.edu/deeprlcourse/.

9 Implementation Tips

In this section, we highlight some useful tips for implementing these models.

Explore Your Data Before starting with steps in the learning phase (see Figure 1), make sure to perform a thorough exploration of your data. What are the results of simple dimensionality reduction methods or clustering? Are the labels reliable? Is there *imbalance* amongst different classes? Are different subpopulations appropriately represented?

Try Simple Methods When starting off with a completely new problem, it is useful to try the simplest version possible. (It might even be worthwhile starting with no learning at all — how does the naive *majority baseline* perform? For datasets with large imbalances, it may be quite strong!) If the dataset is very large, is there some smaller subsampled/downscaled version that can be used for faster preliminary testing? What is the simplest model that might work well? How does a majority baseline perform? (This ties in settings where the data has class imbalance.) Does the model (as expected) overfit to very small subsets of the data?

Where possible, start with well tested models/tasks/methods With the plethora of standard models (many of them pretrained), data augmentation, and optimization methods readily available (Section 3), most new problems will be amenable to some standard set of these choices. *Start with this!* Debugging the dataset and objective function associated with a new problem at the same time as debugging the neural network model, task choice, optimization algorithm, etc is very challenging.

Additionally, many of the standard model/task/method choices are very well benchmarked, and exploring performance in these settings is an excellent first step in understanding the inherent challenges of the new problem. Wherever possible, the easiest way to get starting with the learning phase is to clone an appropriate github repository that has the models and training code needed, and make the minimal edits needed to work with the new dataset and objective function.

First Steps in Debugging Poor Performance Having put together an end-to-end system, you observe that it is not performing well on the validation data. What is the reason? Before getting into more subtle design questions on hyperparameter choice (below), some first things to look at might be (i) Is the model overfitting? If so, more regularization, data augmentation, early stopping, smaller model may help. (ii) Is there a *distribution shift* between the training and validation data? (iii) Is the model underfitting? If so, check the optimization process by seeing if the model *overfits when trained on a smaller subset of the training*

data. Test out a simpler task. Check for noise in the labels or data instances and for distribution shift. (iv) Look at the instances on which the model makes errors. Is there some pattern? For imbalanced datasets, loss function reweighting or more augmentation on the rarer classes can help. (v) How stable is the model performance across multiple random reruns? (vi) What are gradient and intermediate representation norms through the training process?

Which hyperparameters matter most? A big challenge in improving deep learning performance is the multitude of hyperparameters it is possible to change. In practice, some of the simplest hyperparameters often affect performance the most, such as learning rate and learning rate schedule. Picking an optimizer with subtleties such as weight decay correctly implemented can also be very important, see this excellent article on a very popular optimizer, AdamW https://www.fast.ai/2018/07/02/adam-weight-decay/. It might also be very useful to visualize the contributions to total loss from the main objective function vs different regularizers such as weight decay.

Other hyperparameters that can be explored include batch size and data preprocessing, though if standard setups are used for these, varying learning rate related hyperparameters is likely to be the first most useful aspect to explore. To test different hyperparameter settings, it can be very useful to *cross-validate*: hold out a portion of the training data, train different hyperparameter settings on the remaining data, pick whichever hyperparameter setting does best when evaluated on the held out data, and then finally retrain that hyperparameter setting on the full training dataset.

Validate your model thoroughly! Deep learning models are notorious for relying on *spurious correlations* in the data to perform their predictions [8, 162, 247]. By spurious correlation, we mean features in the data instances that happen to co-occur with a specific label, but will not result in a robust, generalizable model. For example, suppose we have data from different chest x-ray machines (corresponding to different hospitals) that we put together to train a deep learning model. It might be the case that one of these machines, so happens to scan many sick patients. The deep learning model might then implicitly learn about the chest x-ray machine instead of the features of the illness. One of the best tests for ensuring the model is learning in a generalizable way is to evaluate the model on data collected *separately* from the training data, which will introduce some natural *distribution shift* and provide a more robust estimate of its accuracy. Some recent interesting papers exploring these questions include [81, 190].

Relatedly, deep neural networks will also pick up on any biases in the data, for example, learning to pay attention to gender (a sensitive attribute) when made to predict age due to class imbalances leading to spurious correlations. This can pose significant challenges for generalizable conclusions in scientific settings where data may be collected from one population, but the predictions must be accurate across all populations. It is therefore important to perform postprocessing analysis on the model representations to identify the presence of such biases. A line of active research studies how to debias these representations [4, 241].

Implementation References Some of the general design considerations when coming to implementation (along with factors affecting larger scale deployment, not explored in this survey) are discussed in this overview https://github.com/chiphuyen/machine-learning-systems-design/blob/master/build/build1/consolidated.pdf.

For specific points on training and debugging deep learning systems, two excellent guides are given by http://josh-tobin.com/assets/pdf/troubleshooting-deep-neural-networks-01-19.pdf and http://karpathy.github.io/2019/04/25/recipe/.

10 Conclusion

As the amount of data collected across many diverse scientific domains continues to increase in both sheer amount and complexity, deep learning methods offer many exciting possibilities for both fundamental predictive problems as well as revealing subtle properties of the underlying data generation process. In this survey, we overviewed many of the highly successful deep learning models, tasks and methodologies, with references to the remarkably comprehensive open-sourced resources developed by the community. We hope that both the overviews and the references serve to accelerate applications of deep learning to many varied scientific problems!

Acknowledgements

The authors would like to thank Jon Kleinberg, Samy Bengio, Yann LeCun, Chiyuan Zhang, Quoc Le, Arun Chaganty, Simon Kornblith, Aniruddh Raghu, John Platt, Richard Murray, Stu Feldman and Guy Gur-Ari for feedback and comments on earlier versions.

References

[1] Waleed Abdulla. Splash of Color: Instance Segmentation with Mask R-CNN and TensorFlow, 2018. https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46.

[2] Roee Aharoni, Melvin Johnson, and Orhan Firat. Massively multilingual neural machine translation. *arXiv preprint arXiv:1903.00089*, 2019.

[3] Jay Alammar. The Illustrated Transformer, 2018. http://jalammar.github.io/illustrated-transformer/.

[4] Mohsan Alvi, Andrew Zisserman, and Christoffer Nellåker. Turning a blind eye: Explicit removal of biases and variation from deep neural network embeddings. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.

[5] Marios Anthimopoulos, Stergios Christodoulidis, Lukas Ebner, Andreas Christe, and Stavroula Mougiakakou. Lung pattern classification for interstitial lung diseases using a deep convolutional neural network. *IEEE transactions on medical imaging*, 35(5):1207–1216, 2016.

[6] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[7] Philip Bachman, R Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. *arXiv preprint arXiv:1906.00910*, 2019.

[8] Marcus A Badgeley, John R Zech, Luke Oakden-Rayner, Benjamin S Glicksberg, Manway Liu, William Gale, Michael V McConnell, Bethany Percha, Thomas M Snyder, and Joel T Dudley. Deep learning predicts hip fracture using confounding patient and healthcare variables. *npj Digital Medicine*, 2(1):31, 2019.

[9] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[10] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[11] Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. End-to-end attention-based large vocabulary speech recognition. In *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4945–4949. IEEE, 2016.

[12] Guha Balakrishnan, Amy Zhao, Mert R Sabuncu, John Guttag, and Adrian V Dalca. An unsupervised learning model for deformable medical image registration. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9252–9260, 2018.

[13] Guha Balakrishnan, Amy Zhao, Mert R Sabuncu, John Guttag, and Adrian V Dalca. Voxelmorph: a learning framework for deformable medical image registration. *IEEE transactions on medical imaging*, 2019.

[14] Joshua Batson and Loic Royer. Noise2self: Blind denoising by self-supervision. *arXiv preprint arXiv:1901.11365*, 2019.

[15] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

[16] Anthony Bau, Yonatan Belinkov, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James Glass. Identifying and controlling important neurons in neural machine translation. *arXiv preprint arXiv:1811.01157*, 2018.

[17] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6541–6549, 2017.

[18] Iz Beltagy, Arman Cohan, and Kyle Lo. Scibert: Pretrained contextualized embeddings for scientific text. *arXiv preprint arXiv:1903.10676*, 2019.

[19] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. Mixmatch: A holistic approach to semi-supervised learning. *arXiv preprint arXiv:1905.02249*, 2019.

[20] Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Larry Jackel, Urs Muller, and Karol Zieba. Visualbackprop: visualizing cnns for autonomous driving. *arXiv preprint arXiv:1611.05418*, 2, 2016.

[21] Xavier Bresson and Thomas Laurent. A two-step graph convolutional decoder for molecule generation. *arXiv preprint arXiv:1906.03412*, 2019.

[22] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[23] Renzhi Cao, Colton Freitas, Leong Chan, Miao Sun, Haiqing Jiang, and Zhangxin Chen. Prolango: protein function prediction using neural machine translation based on a recurrent neural network. *Molecules*, 22(10):1732, 2017.

[24] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields. In *arXiv preprint arXiv:1812.08008*, 2018.

[25] Brandon Carter, Jonas Mueller, Siddhartha Jain, and David Gifford. What made you do this? understanding black-box decisions with sufficient input subsets. *arXiv preprint arXiv:1810.03805*, 2018.

[26] Shan Carter, Zan Armstrong, Ludwig Schubert, Ian Johnson, and Chris Olah. Activation atlas. *Distill*, 2019. https://distill.pub/2019/activation-atlas.

[27] Caroline Chan, Shiry Ginosar, Tinghui Zhou, and Alexei A Efros. Everybody dance now. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5933–5942, 2019.

[28] Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750, 2014.

[29] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

[30] Ting Chen, Yizhou Sun, Yue Shi, and Liangjie Hong. On sampling strategies for neural network-based collaborative filtering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 767–776, 2017.

[31] Yuhua Chen, Yibin Xie, Zhengwei Zhou, Feng Shi, Anthony G Christodoulou, and Debiao Li. Brain mri super resolution using 3d deep densely connected neural networks. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 739–742. IEEE, 2018.

[32] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in neural information processing systems*, pages 577–585, 2015.

[33] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International conference on medical image computing and computer-assisted intervention*, pages 424–432. Springer, 2016.

[34] Kevin Clark, Minh-Thang Luong, Christopher D Manning, and Quoc V Le. Semi-supervised sequence modeling with cross-view training. *arXiv preprint arXiv:1809.08370*, 2018.

[35] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999, 2016.

[36] Taco S Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical cnns. *arXiv preprint arXiv:1801.10130*, 2018.

[37] Peter Corbett and John Boyle. Improving the learning of chemical-protein interactions from literature using transfer learning and specialized word embeddings. *Database*, 2018, 2018.

[38] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical data augmentation with no separate search. *arXiv preprint arXiv:1909.13719*, 2019.

[39] Adrian Dalca, Marianne Rakic, John Guttag, and Mert Sabuncu. Learning conditional deformable templates with convolutional networks. In *Advances in neural information processing systems*, pages 804–816, 2019.

[40] Yann Dauphin. mixup: Beyond Empirical Risk Minimization Image, 2017. https://www.dauphin.io/.

[41] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

[42] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[43] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[44] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[45] Yiming Ding, Jae Ho Sohn, Michael G Kawczynski, Hari Trivedi, Roy Harnish, Nathaniel W Jenkins, Dmytro Lituiev, Timothy P Copeland, Mariam S Aboian, Carina Mari Aparici, et al. A deep learning model to predict a diagnosis of alzheimer disease by using 18f-fdg pet of the brain. *Radiology*, 290(2):456–464, 2018.

[46] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[47] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[48] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430, 2015.

[49] Jeff Donahue and Karen Simonyan. Large scale adversarial representation learning. In *Advances in Neural Information Processing Systems*, pages 10541–10551, 2019.

[50] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2015.

[51] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. In *Advances in neural information processing systems*, pages 766–774, 2014.

[52] Yong Du, Wei Wang, and Liang Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1110–1118, 2015.

[53] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[54] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale. *arXiv preprint arXiv:1808.09381*, 2018.

[55] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.

[56] Linjing Fang, Fred Monroe, Sammy Weiser Novak, Lyndsey Kirk, Cara R Schiavon, B Yu Seungyoon, Tong Zhang, Melissa Wu, Kyle Kastner, Yoshiyuki Kubota, et al. Deep learning-based point-scanning super-resolution imaging. *bioRxiv*, page 740548, 2019.

[57] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016.

[58] Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. *arXiv preprint arXiv:2002.12880*, 2020.

[59] Ruth C Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3429–3437, 2017.

[60] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems*, pages 6530–6539, 2017.

[61] Ferenc Galkó and Carsten Eickhoff. Biomedical question answering via weighted neural network passage retrieval. In *European Conference on Information Retrieval*, pages 523–528. Springer, 2018.

[62] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. *arXiv preprint arXiv:1409.7495*, 2014.

[63] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[64] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[65] Amirata Ghorbani, Vivek Natarajan, David Coz, and Yuan Liu. Dermgan: Synthetic generation of clinical skin images with pathology. *arXiv preprint arXiv:1911.08716*, 2019.

[66] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

[67] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[68] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[69] Priya Goyal, Dhruv Mahajan, Abhinav Gupta, and Ishan Misra. Scaling and benchmarking self-supervised visual representation learning. *arXiv preprint arXiv:1905.01235*, 2019.

[70] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[71] Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. *arXiv preprint arXiv:1803.10459*, 2018.

[72] Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22):2402–2410, 2016.

[73] Maryam Habibi, Leon Weber, Mariana Neves, David Luis Wiegandt, and Ulf Leser. Deep learning with word embeddings improves biomedical named entity recognition. *Bioinformatics*, 33(14):i37–i48, 2017.

[74] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *Advances in neural information processing systems*, pages 8527–8537, 2018.

[75] Ankur Handa, Michael Bloesch, Viorica Pătrăucean, Simon Stent, John McCormac, and Andrew Davison. gvnn: Neural network library for geometric computer vision. In *European Conference on Computer Vision*, pages 67–82. Springer, 2016.

[76] Boris Hanin. Which neural net architectures give rise to exploding and vanishing gradients? In *Advances in Neural Information Processing Systems*, pages 582–591, 2018.

[77] Jack Hanson, Yuedong Yang, Kuldip Paliwal, and Yaoqi Zhou. Improving protein disorder prediction by deep bidirectional long short-term memory recurrent neural networks. *Bioinformatics*, 33(5):685–692, 2016.

[78] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[79] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[80] Rhys Heffernan, Yuedong Yang, Kuldip Paliwal, and Yaoqi Zhou. Capturing non-local interactions by long short-term memory bidirectional recurrent neural networks for improving prediction of protein secondary structure, backbone angles, contact numbers and solvent accessibility. *Bioinformatics*, 33(18):2842–2849, 2017.

[81] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

[82] Dan Hendrycks, Norman Mu, Ekin D Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. *arXiv preprint arXiv:1912.02781*, 2019.

[83] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in neural information processing systems*, pages 1693–1701, 2015.

[84] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

[85] Daniel Ho, Eric Liang, Ion Stoica, Pieter Abbeel, and Xi Chen. Population based augmentation: Efficient learning of augmentation policy schedules. *arXiv preprint arXiv:1905.05393*, 2019.

[86] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *arXiv preprint arXiv:1902.00275*, 2019.

[87] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[88] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[89] Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S Weld. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 541–550. Association for Computational Linguistics, 2011.

[90] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly.

Parameter-efficient transfer learning for nlp. *arXiv preprint arXiv:1902.00751*, 2019.

[91] Andrew G Howard. Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv:1312.5402*, 2013.

[92] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[93] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

[94] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[95] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

[96] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[97] Na Ji. Adaptive optical fluorescence microscopy. *Nature methods*, 14(4):374, 2017.

[98] Robin Jia and Percy Liang. Data recombination for neural semantic parsing. *arXiv preprint arXiv:1606.03622*, 2016.

[99] Matthew Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in neural information processing systems*, pages 2946–2954, 2016.

[100] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[101] MF Kasim, D Watson-Parris, L Deaconu, S Oliver, P Hatfield, DH Froula, G Gregori, M Jarvis, S Khatiwala, J Korenaga, et al. Up to two billion times acceleration of scientific simulations with deep neural architecture search. *arXiv preprint arXiv:2001.08055*, 2020.

[102] Jeremy Kawahara, Sara Daneshvar, Giuseppe Argenziano, and Ghassan Hamarneh. Seven-point checklist and skin lesion classification using multitask multimodal neural nets. *IEEE journal of biomedical and health informatics*, 23(2):538–546, 2018.

[103] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

[104] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). *arXiv preprint arXiv:1711.11279*, 2017.

[105] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un) reliability of saliency methods. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 267–280. Springer, 2019.

[106] Pieter-Jan Kindermans, Kristof T Schütt, Maximilian Alber, Klaus-Robert Müller, Dumitru Erhan, Been Kim, and Sven Dähne. Learning how to explain neural networks: Patternnet and patternattribution. *arXiv preprint arXiv:1705.05598*, 2017.

[107] D Kingma, Tim Salimans, R Josefowicz, Xi Chen, Ilya Sutskever, Max Welling, et al. Improving variational autoencoders with inverse autoregressive flow. 2017.

[108] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.

[109] Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pages 3581–3589, 2014.

[110] Sosuke Kobayashi. Contextual augmentation: Data augmentation by words with paradigmatic relations. *arXiv preprint arXiv:1805.06201*, 2018.

[111] Kaname Kojima, Shu Tadaka, Fumiki Katsuoka, Gen Tamiya, Masayuki Yamamoto, and Kengo Kinoshita. A recurrent neural network based method for genotype imputation on phased genotype data. *bioRxiv*, page 821504, 2019.

[112] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning. *arXiv preprint arXiv:1901.09005*, 2019.

[113] Patrick T Komiske, Eric M Metodiev, and Jesse Thaler. Energy flow networks: deep sets for particle jets. *Journal of High Energy Physics*, 2019(1):121, 2019.

[114] Shu Kong and Charless Fowlkes. Image reconstruction with predictive filter flow. *arXiv preprint arXiv:1811.11482*, 2018.

[115] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. *arXiv preprint arXiv:1905.00414*, 2019.

[116] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2661–2671, 2019.

[117] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[118] Sneha Reddy Kudugunta, Ankur Bapna, Isaac Caswell, Naveen Arivazhagan, and Orhan Firat. Investigating multilingual nmt representations at scale. *arXiv preprint arXiv:1909.02197*, 2019.

[119] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*, 2016.

[120] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML*, volume 3, page 2, 2013.

[121] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: pre-trained biomedical language representation model for biomedical text mining. *arXiv preprint arXiv:1901.08746*, 2019.

[122] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2noise: Learning image restoration without clean data. *arXiv preprint arXiv:1803.04189*, 2018.

[123] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

[124] Chaolong Li, Zhen Cui, Wenming Zheng, Chunyan Xu, and Jian Yang. Spatio-temporal graph convolution for skeleton based action recognition. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[125] Hailiang Li, Jian Weng, Yujian Shi, Wanrong Gu, Yijun Mao, Yonghua Wang, Weiwei Liu, and Jiajie Zhang. An improved deep learning approach for detection of thyroid papillary cancer in ultrasound images. *Scientific reports*, 8(1):6600, 2018.

[126] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John E Hopcroft. Convergent learning: Do different neural networks learn the same representations? In *Iclr*, 2016.

[127] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[128] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[129] Fang Liu, Zhaoye Zhou, Hyungseok Jang, Alexey Samsonov, Gengyan Zhao, and Richard Kijowski. Deep convolutional neural network and 3d deformable approach for tissue segmentation in musculoskeletal magnetic resonance imaging. *Magnetic resonance in medicine*, 79(4):2379–2391, 2018.

[130] Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

[131] Shengyu Liu, Buzhou Tang, Qingcai Chen, and Xiaolong Wang. Effects of semantic features on machine learning-based drug name recognition systems: word embeddings vs. manually constructed dictionaries. *Information*, 6(4):848–865, 2015.

[132] Xueliang Liu. Deep recurrent neural network for protein function prediction from sequence. *arXiv preprint arXiv:1701.08318*, 2017.

[133] Yao Liu, Omer Gottesman, Aniruddh Raghu, Matthieu Komorowski, Aldo A Faisal, Finale Doshi-Velez, and Emma Brunskill. Representation balancing mdps for off-policy policy evaluation. In *Advances in Neural Information Processing Systems*, pages 2644–2653, 2018.

[134] Yao Liu, Adith Swaminathan, Alekh Agarwal, and Emma Brunskill. Off-policy policy gradient with state distribution correction. *arXiv preprint arXiv:1904.08473*, 2019.

[135] Yun Liu, Krishna Gadepalli, Mohammad Norouzi, George E Dahl, Timo Kohlberger, Aleksey Boyko, Subhashini Venugopalan, Aleksei Timofeev, Philip Q Nelson, Greg S Corrado, et al. Detecting cancer metastases on gigapixel pathology images. *arXiv preprint arXiv:1703.02442*, 2017.

[136] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[137] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Deep transfer learning with joint adaptation networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2208–2217. JMLR. org, 2017.

[138] Romain Lopez, Jeffrey Regier, Michael Cole, Michael Jordan, and Nir Yosef. A deep generative model for gene expression profiles from single-cell rna sequencing. *arXiv preprint arXiv:1709.02082*, 2017.

[139] Donghuan Lu, Karteek Popuri, Gavin Weiguang Ding, Rakesh Balachandar, and Mirza Faisal Beg. Multimodal and multiscale deep neural networks for the early diagnosis of alzheimer's disease using structural mr and fdg-pet images. *Scientific reports*, 8(1):5697, 2018.

[140] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774, 2017.

[141] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[142] Ali Madani, Bryan McCann, Nikhil Naik, Nitish Shirish Keskar, Namrata Anand, Raphael R Eguchi, Possu Huang, and Richard Socher. Progen: Language modeling for protein generation. *bioRxiv*, 2020.

[143] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 181–196, 2018.

[144] Hongzi Mao, Parimarjan Negi, Akshay Narayan, Hanrui Wang, Jiacheng Yang, Haonan Wang, Ryan Marcus, Ravichandra Addanki, Mehrdad Khani, Songtao He, et al. Park: An open platform for learning augmented computer systems. 2019.

[145] Daniel L Marino, Kasun Amarasinghe, and Milos Manic. Building energy load forecasting using deep neural networks. In *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 7046–7051. IEEE, 2016.

[146] Alexander Mathis, Pranav Mamidanna, Kevin M Cury, Taiga Abe, Venkatesh N Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature neuroscience*, 21(9):1281, 2018.

[147] Mackenzie Weygandt Mathis and Alexander Mathis. Deep learning tools for the measurement of animal behavior in neuroscience. *Current Opinion in Neurobiology*, 60:1–11, 2020.

[148] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[149] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and Optimizing LSTM Language Models. *arXiv preprint arXiv:1708.02182*, 2017.

[150] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. An Analysis of Neural Language Modeling at Multiple Scales. *arXiv preprint arXiv:1803.08240*, 2018.

[151] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[152] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[153] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics, 2009.

[154] Ishan Misra and Laurens van der Maaten. Self-supervised learning of pretext-invariant representations, 2019.

[155] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2018.

[156] Pim Moeskops, Max A Viergever, Adriënne M Mendrik, Linda S de Vries, Manon JNL Benders, and Ivana Išgum. Automatic segmentation of mr brain images with a convolutional neural network. *IEEE transactions on medical imaging*, 35(5):1252–1261, 2016.

[157] Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *Advances in Neural Information Processing Systems*, pages 5727–5736, 2018.

[158] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016.

[159] Jiquan Ngiam, Daiyi Peng, Vijay Vasudevan, Simon Kornblith, Quoc V Le, and Ruoming Pang. Domain adaptive transfer learning with specialist models. *arXiv preprint arXiv:1811.07056*, 2018.

[160] Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. Interpretml: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223*, 2019.

[161] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pages 69–84. Springer, 2016.

[162] Luke Oakden-Rayner, Jared Dunnmon, Gustavo Carneiro, and Christopher Ré. Hidden stratification causes clinically meaningful failures in machine learning for medical imaging. *arXiv preprint arXiv:1909.12475*, 2019.

[163] Chris Olah. Understanding LSTM Networks, 2015. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[164] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

[165] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[166] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. https://distill.pub/2018/building-blocks.

[167] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[168] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[169] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[170] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2, 2012.

[171] Deepak Pathak, Philipp Krahenbuhl, and Trevor Darrell. Constrained convolutional neural networks for weakly supervised segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1796–1804, 2015.

[172] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

[173] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[174] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[175] Gianluca Pollastri, Darisz Przybylski, Burkhard Rost, and Pierre Baldi. Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. *Proteins: Structure, Function, and Bioinformatics*, 47(2):228–235, 2002.

[176] Ryan Poplin, Avinash V Varadarajan, Katy Blumer, Yun Liu, Michael V McConnell, Greg S Corrado, Lily Peng, and Dale R Webster. Prediction of cardiovascular risk factors from retinal fundus photographs via deep learning. *Nature Biomedical Engineering*, 2(3):158, 2018.

[177] Rory M Power and Jan Huisken. A guide to light-sheet fluorescence microscopy for multiscale imaging. *Nature methods*, 14(4):360, 2017.

[178] Doina Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

[179] Siyuan Qiao, Wei Shen, Zhishuai Zhang, Bo Wang, and Alan Yuille. Deep co-training for semi-supervised image recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 135–152, 2018.

[180] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019.

[181] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[182] Aniruddh Raghu, Matthieu Komorowski, Leo Anthony Celi, Peter Szolovits, and Marzyeh Ghassemi. Continuous state-space models for optimal sepsis treatment-a deep reinforcement learning approach. *arXiv preprint arXiv:1705.08422*, 2017.

[183] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in Neural Information Processing Systems*, pages 6076–6085, 2017.

[184] Maithra Raghu, Chiyuan Zhang, Jon Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning for medical imaging. In *Advances in Neural Information Processing Systems*, pages 3342–3352, 2019.

[185] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding, Aarti Bagul, Curtis Langlotz, Katie Shpanskaya, et al. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv:1711.05225*, 2017.

[186] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[187] Bharath Ramsundar, Steven Kearnes, Patrick Riley, Dale Webster, David Konerding, and Vijay Pande. Massively multitask networks for drug discovery. *arXiv preprint arXiv:1502.02072*, 2015.

[188] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282, 2017.

[189] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *arXiv preprint arXiv:1906.00446*, 2019.

[190] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? *arXiv preprint arXiv:1902.10811*, 2019.

[191] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[192] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[193] Donatas Repecka, Vykintas Jauniskis, Laurynas Karpus, Elzbieta Rembeza, Jan Zrimec, Simona Poviloniene, Irmantas Rokaitis, Audrius Laurynenas, Wissam Abuajwa, Otto Savolainen, et al. Expanding functional protein sequence space using generative adversarial networks. *bioRxiv*, page 789719, 2019.

[194] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.

[195] Alexander Rives, Siddharth Goyal, Joshua Meier, Demi Guo, Myle Ott, C Lawrence Zitnick, Jerry Ma, and Rob Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv*, page 622803, 2019.

[196] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[197] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.

[198] Andrei A Rusu, Mel Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016.

[199] Ruhan Sa, William Owens, Raymond Wiegand, Mark Studin, Donald Capoferri, Kenneth Barooha, Alexander Greaux, Robert Rattray, Adam Hutton, John Cintineo, et al. Intervertebral disc detection in x-ray images using faster r-cnn. In *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 564–567. IEEE, 2017.

[200] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

[201] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[202] Saman Sarraf, Ghassem Tofighi, et al. Deepad: Alzheimer disease classification via deep convolutional neural networks using mri and fmri. *BioRxiv*, page 070441, 2016.

[203] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[204] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 618–626, 2017.

[205] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, pages 1–5, 2020.

[206] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*, 2015.

[207] Lloyd S Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317, 1953.

[208] Jianghong Shi, Eric Shea-Brown, and Michael Buice. Comparison against task driven artificial neural networks reveals functional properties in mouse visual cortex. In *Advances in Neural Information Processing Systems*, pages 5765–5775, 2019.

[209] Susan M Shortreed, Eric Laber, Daniel J Lizotte, T Scott Stroup, Joelle Pineau, and Susan A Murphy. Informing sequential clinical decision-making through reinforcement learning: an empirical study. *Machine learning*, 84(1-2):109–136, 2011.

[210] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3145–3153. JMLR. org, 2017.

[211] Rui Shu, Hung H Bui, Hirokazu Narui, and Stefano Ermon. A dirt-t approach to unsupervised domain adaptation. *arXiv preprint arXiv:1802.08735*, 2018.

[212] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[213] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[214] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[215] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[216] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Advances in neural information processing systems*, pages 3738–3746, 2016.

[217] Youyi Song, Ling Zhang, Siping Chen, Dong Ni, Baopu Li, Yongjing Zhou, Baiying Lei, and Tianfu Wang. A deep learning based framework for accurate segmentation of cervical cytoplasm and nuclei. In *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 2903–2906. IEEE, 2014.

[218] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5693–5703, 2019.

[219] Yu Sun, Eric Tzeng, Trevor Darrell, and Alexei A Efros. Unsupervised domain adaptation through self-supervision. *arXiv preprint arXiv:1909.11825*, 2019.

[220] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3319–3328. JMLR. org, 2017.

[221] I Sutskever, O Vinyals, and QV Le. Sequence to sequence learning with neural networks. *Advances in NIPS*, 2014.

[222] Ryo Takahashi, Takashi Matsubara, and Kuniaki Uehara. Data augmentation using random image cropping and patching for deep cnns. *IEEE Transactions on Circuits and Systems for Video Technology*, 2019.

[223] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[224] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. *arXiv preprint arXiv:1911.09070*, 2019.

[225] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in neural information processing systems*, pages 1195–1204, 2017.

[226] Dimitry Tegunov and Patrick Cramer. Real-time cryo-em data pre-processing with warp. *BioRxiv*, page 338558, 2018.

[227] Yee Liang Thian, Yiting Li, Pooja Jagmohan, David Sia, Vincent Ern Yao Chan, and Robby T Tan. Convolutional neural networks for automated fracture detection and localization on wrist radiographs. *Radiology: Artificial Intelligence*, 1(1):e180001, 2019.

[228] Eric J Topol. High-performance medicine: the convergence of human and artificial intelligence. *Nature medicine*, 25(1):44–56, 2019.

[229] Raphael Townshend, Rishi Bedi, Patricia Suriana, and Ron Dror. End-to-end learning on 3d protein structure for interface prediction. In *Advances in Neural Information Processing Systems*, pages 15616–15625, 2019.

[230] Vahe Tshitoyan, John Dagdelen, Leigh Weston, Alexander Dunn, Ziqin Rong, Olga Kononova, Kristin A Persson, Gerbrand Ceder, and Anubhav Jain. Unsupervised word embeddings capture latent knowledge from materials science literature. *Nature*, 571(7763):95–98, 2019.

[231] Kensuke Umehara, Junko Ota, and Takayuki Ishida. Application of super-resolution convolutional neural

network for enhancing image resolution in chest ct. *Journal of digital imaging*, 31(4):441–450, 2018.

[232] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pages 4790–4798, 2016.

[233] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[234] Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

[235] Elena Voita, Rico Sennrich, and Ivan Titov. The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives. *arXiv preprint arXiv:1909.01380*, 2019.

[236] Christian Wachinger, Martin Reuter, and Tassilo Klein. Deepnat: Deep convolutional neural network for segmenting neuroanatomy. *NeuroImage*, 170:434–445, 2018.

[237] Kun Wang, Bite Yang, Guohai Xu, and Xiaofeng He. Medical question retrieval based on siamese neural network and transfer learning method. In *International Conference on Database Systems for Advanced Applications*, pages 49–64. Springer, 2019.

[238] Nancy XR Wang, Ali Farhadi, Rajesh PN Rao, and Bingni W Brunton. Ajile movement prediction: Multimodal deep learning for natural human neural recordings and video. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[239] William Yang Wang and Diyi Yang. That's so annoying!!!: A lexical and frame-semantic embedding based data augmentation approach to automatic categorization of annoying behaviors using# petpeeve tweets. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2557–2563, 2015.

[240] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7794–7803, 2018.

[241] Zeyu Wang, Klint Qinami, Yannis Karakozis, Kyle Genova, Prem Nair, Kenji Hata, and Olga Russakovsky. Towards fairness in visual recognition: Effective strategies for bias mitigation. *arXiv preprint arXiv:1911.11834*, 2019.

[242] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

[243] Jason W Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.

[244] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4724–4732, 2016.

[245] Martin Weigert, Uwe Schmidt, Tobias Boothe, Andreas Müller, Alexandr Dibrov, Akanksha Jain, Benjamin Wilhelm, Deborah Schmidt, Coleman Broaddus, Siân Culley, et al. Content-aware image restoration: pushing the limits of fluorescence microscopy. *Nature methods*, 15(12):1090, 2018.

[246] David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. Structured training for neural network transition-based parsing. *arXiv preprint arXiv:1506.06158*, 2015.

[247] Julia K Winkler, Christine Fink, Ferdinand Toberer, Alexander Enk, Teresa Deinlein, Rainer Hofmann-Wellenhof, Luc Thomas, Aimilios Lallas, Andreas Blum, Wilhelm Stolz, et al. Association between surgical skin markings in dermoscopic images and diagnostic performance of a deep learning convolutional neural network for melanoma recognition. *JAMA dermatology*, 155(10):1135–1141, 2019.

[248] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[249] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

[250] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[251] Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Unsupervised data augmentation. *arXiv preprint arXiv:1904.12848*, 2019.

[252] Qizhe Xie, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Self-training with noisy student improves imagenet classification. *arXiv preprint arXiv:1911.04252*, 2019.

[253] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[254] Jun Xu, Xiaofei Luo, Guanhao Wang, Hannah Gilmore, and Anant Madabhushi. A deep convolutional neural network for segmenting and classifying epithelial and stromal regions in histopathological images. *Neurocomputing*, 191:214–223, 2016.

[255] Xueting Yan, Ishan Misra, Abhinav Gupta, Deepti Ghadiyaram, and Dhruv Mahajan. Clusterfit: Improving generalization of visual representations. *arXiv preprint arXiv:1912.03330*, 2019.

[256] Yilong Yang, Zhuyifan Ye, Yan Su, Qianqian Zhao, Xiaoshan Li, and Defang Ouyang. Deep learning for in vitro prediction of pharmaceutical formulations. *Acta pharmaceutica sinica B*, 9(1):177–185, 2019.

[257] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

[258] Koichiro Yasaka, Hiroyuki Akai, Osamu Abe, and Shigeru Kiryu. Deep learning with convolutional neural network for differentiation of liver masses at dynamic contrast-enhanced ct: a preliminary study. *Radiology*, 286(3):887–896, 2017.

[259] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[260] Yuhui Yuan, Xilin Chen, and Jingdong Wang. Object-contextual representations for semantic segmentation. *arXiv preprint arXiv:1909.11065*, 2019.

[261] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6023–6032, 2019.

[262] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

[263] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[264] Daojian Zeng, Kang Liu, Yubo Chen, and Jun Zhao. Distant supervision for relation extraction via piecewise convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1753–1762, 2015.

[265] Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S4l: Self-supervised semi-supervised learning. *arXiv preprint arXiv:1905.03670*, 2019.

[266] Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, et al. The visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*, 2019.

[267] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. Dive into deep learning. *Unpublished draft. Retrieved*, 3:319, 2019.

[268] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[269] Junkang Zhang, Haigen Hu, Shengyong Chen, Yujiao Huang, and Qiu Guan. Cancer cells detection in phase-contrast microscopy images based on faster r-cnn. In *2016 9th International Symposium on Computational Intelligence and Design (ISCID)*, volume 1, pages 363–367. IEEE, 2016.

[270] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2472–2481, 2018.

[271] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

[272] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*, 2014.

[273] Qingyu Zhou, Nan Yang, Furu Wei, Shaohan Huang, Ming Zhou, and Tiejun Zhao. Neural document summarization by jointly learning to score and select sentences. *arXiv preprint arXiv:1807.02305*, 2018.

[274] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[275] Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. *arXiv preprint arXiv:1702.04595*, 2017.